ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación
Maestría en Sistemas de Información Gerencial

"Elaboración de una metodología para el desarrollo de aplicaciones Web en la Superintendencia de Bancos y Seguros"

TESIS DE GRADO

Previa a la obtención del Título de:

MAGÍSTER EN SISTEMAS DE INFORMACIÓN GERENCIAL

Presentado por:

LSI. William Luis Cargua Freire

Guayaquil - Ecuador

Año

2009

AGRADECIMIENTO

Agradezco a Dios, por permitirme llegar hasta donde estoy y por bendecirme con la vida.

A mi familia, que siempre me dieron su apoyo durante la elaboración de mi tesis, a mi padre el Sr. Luis Cargua Chicaiza, a mi madre la Lcda. Nancy Freire Arévalo y a mi hermano el LSI. Rubén Cargua Freire.

A mi Director de Tesis, el Ing. Lenín Freire Cobo, por su valiosa ayuda.

A mi casa de estudios, la ESPOL, que ha abierto mundos infinitos de conocimiento para mi desarrollo intelectual, en donde he aprendido que el universo es infinito y que Yo soy parte de él.

DEDICATORIA

A **DIOS**

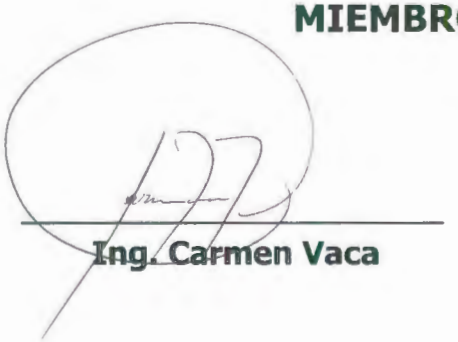
A **MIS PADRES**

A **MI HERMANO**

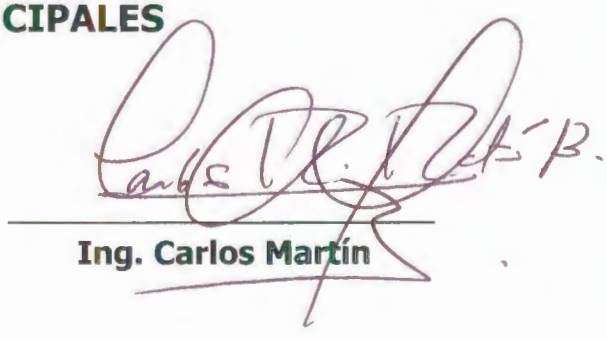
TRIBUNAL DE GRADUACIÓN

Ing. Lenín Freire Cobo
Director de tesis

MIEMBROS PRINCIPALES



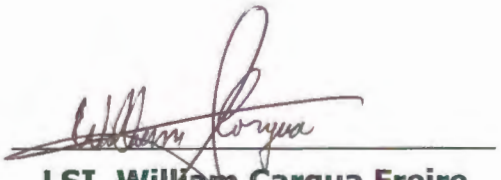
Ing. Carmen Vaca



Ing. Carlos Martín

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, me corresponde exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".



LSI. William Cargua Freire

RESUMEN

En esta tesis se aborda la problemática que implica para los profesionales de sistemas la comprensión (entendimiento) del negocio, integrado a la solución tecnológica de soporte. Todas las metodologías de desarrollo de proyectos software dan por entendido las actividades del negocio de una empresa, pero no proporcionan una fase que permita al profesional en sistemas comprender el entorno en el que va a trabajar. La sistematización de esta información permite entender en qué medida el proyecto de software es un medio para alcanzar los objetivos de la empresa.

Este proyecto formula la propuesta de una metodología de desarrollo de aplicaciones Web que a su vez intenta complementar las metodologías de desarrollo existentes, mediante la utilización de nuevas técnicas.

En relación a esto se describe en el **Capítulo I** las definiciones de los principales conceptos relacionados con las técnicas y guías que conforman a las metodologías de desarrollo que existen en la actualidad.

Se proporciona en el **Capítulo II** la descripción de cómo se realiza en la actualidad el proceso de desarrollo de aplicaciones en la Superintendencia de bancos y seguros, destacando cuáles son los problemas y dificultades encontradas en cada parte del proceso.

Se presenta en el **Capítulo III** el detalle de la solución propuesta para los problemas existentes en el proceso de desarrollo de aplicaciones en la Superintendencia de bancos y seguros. Esta solución consiste en promover una metodología para el desarrollo de aplicaciones que sugiere la utilización de estándares, formatos de formularios y documentos, la definición de las diferentes fases que componen la metodología, sus ventajas, desventajas y la evaluación de cada una de ellas.

Luego en el **Capítulo IV** se describe la forma de capacitación a los profesionales de sistemas, para utilizar la metodología de desarrollo, proporcionando para ello los escenarios de aplicación de la metodología sobre algunos casos de estudio.

Después en este proyecto se presenta en el **Capítulo V** la comparación de la metodología de desarrollo propuesta con otras metodologías de desarrollo orientadas a objetos, mostrando cuáles son las ventajas y desventajas entre utilizar una u otra metodología.

Finalmente en el **Capítulo VI** se describe el desarrollo de un prototipo de software, basando su desarrollo en la metodología propuesta, mostrando los resultados de su aplicación.

ÍNDICE GENERAL

Pág.

ÍNDICE DE FIGURAS

Pág.

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO VI

ÍNDICE DE TABLAS

Pág.

CAPÍTULO III

CAPÍTULO V

INTRODUCCIÓN

El propósito de este proyecto de tesis es la elaboración de una metodología para el desarrollo de aplicaciones Web en la Superintendencia de Bancos y Seguros. Existen en la actualidad tecnologías que permiten un rápido desarrollo de aplicaciones poco reusables y difíciles de mantener. La metodología propuesta en este proyecto, aplicada con las tecnologías brevemente descritas por cada etapa del desarrollo, permitirá obtener aplicaciones mediante un proceso de desarrollo en capas, aprovechando al máximo la potencia de la programación orientada a objetos.

Mediante la metodología propuesta se podrá especificar el desarrollo de aplicaciones para ambientes Web, extendiendo algunos conceptos sobre los métodos Orientados a Objetos existentes. Las aplicaciones Web tienen una base común con las aplicaciones tradicionales: la funcionalidad de la aplicación y la interacción con los usuarios. Sin embargo, introducen nuevas características navegacionales que deben ser capturadas para representar de

una manera más precisa y aproximada a la aplicación. Es por ello que dentro del marco de la metodología propuesta se establecen las técnicas necesarias para la administración de las características navegacionales de las aplicaciones Web de la Superintendencia de Bancos y Seguros.

OBJETIVOS ESPECÍFICOS

- Identificar las últimas tendencias en desarrollo Web.

- Establecer el Proceso de desarrollo, indicando cuáles son las etapas del ciclo de vida de desarrollo de una aplicación Web.

- Explicar en qué consiste el desarrollo Web centrado en el usuario/a.

- Elaborar técnicas de requisitos que permitan identificar el nivel de la complejidad de una aplicación Web, para en cada etapa del desarrollo tener el criterio de selección de que técnicas y guías son opcionales e imprescindibles de utilizar.

- Describir qué es el diseño orientado a objetos y qué facilidades aporta al desarrollo Web.

- Se especificarán las técnicas y guías por cada etapa de desarrollo, mediante la documentación de las actividades que se deben seguir por etapa, estableciendo estándares de diagramas técnicos, codificación, visuales, de navegabilidad, de pruebas e implementación, manuales técnicos de desarrollo, manuales de usuario, y manuales de instalación y configuración de las aplicaciones. Las técnicas y guías propuestas se basarán en UML, en tecnología XML y de Base de Datos Relacional.

- Definir las técnicas de medición de la usabilidad o facilidad de uso de las aplicaciones Web.

- Establecer el uso de un modelo de arquitectura para el funcionamiento de las aplicaciones Web.

- Seleccionar un marco de trabajo (*framework*) de entre los más utilizados en la actualidad, obteniendo de esta forma aplicaciones Web eficientes, de calidad y mantenibles, reduciendo al mismo tiempo el coste y la duración del desarrollo.

ALCANCE DEL PROYECTO

- Definir la metodología de desarrollo de aplicaciones Web para la Superintendencia de Bancos y Seguros.

- Los estándares de desarrollo se basarán en UML, en tecnología XML, Base de Datos Relacional y los componentes de la arquitectura J2EE, los que servirán como guía en el desarrollo de aplicaciones Web.

- Permitir a los administradores de sistemas incorporar políticas para el desarrollo de aplicaciones Web, basándose en la metodología propuesta.

- Proveer la metodología de desarrollo, para que sirva como una herramienta en la organización de los procesos internos del desarrollo de aplicaciones en la Superintendencia de Bancos y Seguros.

- Capacitar a los jefes de proyectos, arquitectos de software, usuarios desarrolladores de la Superintendencia de Bancos y Seguros, en el uso de la metodología de desarrollo.

- Aplicar la metodología de desarrollo a un proyecto mediante la construcción de un prototipo de software.

CAPÍTULO I

"MARCO CONCEPTUAL."

CAPÍTULO I

1. MARCO CONCEPTUAL

1.1. Antecedentes y definiciones de las metodologías de desarrollo

Los seres humanos vivimos en un universo caracterizado por su variedad de propiedades, conexiones y eventos. Esta amplia gama de impresiones reguladas por un principio de organización dio origen al concepto conocido como método, un ente encargado de mantener en orden la realidad que perciben los sentidos. A través de la historia, distintos pensadores y filósofos de la humanidad han tratado de seguir métodos en cada una de las actividades del conocimiento tales como las matemáticas, la biología, la astronomía, la economía y en el último siglo: las ciencias de la computación.

Todo este conocimiento dio origen al concepto de metodología: un

sistema de principios y normas generales de organización y estructuración teórico-práctica de actividades. **[1]** Las metodologías siempre han existido en nuestra historia, desde que el hombre pensó en los primeros métodos para desarrollar sus habilidades físicas e intelectuales para así poder dominar la naturaleza, hasta cuando empezó a generar estrategias de comunicación para realizar procesos de trabajo más eficientes y eficaces.

1.1.1. Las metodologías de desarrollo de software: la punta del iceberg de la evolución de la filosofía del conocimiento

En el siglo XX, se dio una de las más grandes revoluciones de la humanidad: la revolución informática. A medida que la electrónica evolucionaba, los diseños de sistemas computacionales realizados en el siglo XIX se hacían cada vez más reales. Primero eran computadores análogos, luego gracias al uso de la electrónica digital (Que inventó Claude Shannon en 1937) se pudieron crear los primeros computadores con la capacidad de ser programados. A pesar que no se sabe con certeza cuándo ni dónde se construyó el primer computador, se destacan casos como el computador

Atanasoff Berry, el ENIAC (1943), el computador British Colossus (1944) y las máquinas Konrad Zuse's Z.

De esta manera se comenzó a usar el término Ingeniería de Software para referenciar a la profesión encargada de crear y mantener aplicaciones software por medio del uso de tecnologías, prácticas y métodos para gestionar un proyecto relacionado con las ciencias de la computación **[4]**.

Como se presenta en el libro de Roger PRESSMAN "Ingeniería del Software-Un enfoque práctico" **[5]**, el software de código abierto y/o libre, ha ido evolucionando a medida que lo ha hecho su experiencia como organización.

Nos encontramos en la era del pensamiento creativo. "Cada cerebro humano es una gran cámara de creatividad que necesita de las herramientas con las cuales su imaginación pueda ser liberada"**[6]**.

1.2. Las metodologías de desarrollo actuales

¿Qué metodología debo usar para el desarrollo de un Software? Todos en algún momento nos hemos hecho esta pregunta, cuando hemos tenido que desarrollar un software. Y de hecho esta pregunta se torna muy importante, pues como arquitectos de Software, debemos tener un plano en que apoyarnos.

1.2.1. Metodologías tradicionales en el desarrollo de software

Se caracterizan por exponer procesos basados en planeación exhaustiva. Esta planeación se realiza esperando que el resultado de cada proceso sea determinante y predecible. La experiencia ha mostrado que, como consecuencia de las características del software, los resultados de los procesos no son siempre predecibles y sobre todo, es difícil predecir desde el comienzo del proyecto cada resultado.

1.2.1.1. El modelo de cascada

Remontándose a la historia, **el modelo de cascada** fue uno de los primeros modelos de ciclo de vida (MCV) que formalizó un conjunto de procesos de desarrollo de software. Este MCV describe un orden secuencial en la ejecución de los procesos asociados.

Figura 1-1 El modelo de desarrollo en cascada

Desde el punto de vista de Ingeniería de Software el modelo en cascada, es el enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma tal que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.

1.2.1.2. El modelo espiral

El modelo espiral se postuló como una alternativa al modelo de cascada. La ventaja de este modelo radica en el perfeccionamiento de las soluciones encontradas con cada ciclo de desarrollo, en términos de dar respuesta a los requerimientos inicialmente analizados.

Las actividades de este modelo son una espiral, cada bucle es una actividad. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior. Para cada actividad habrá cuatro tareas que se pueden visualizar en la figura 1-2.

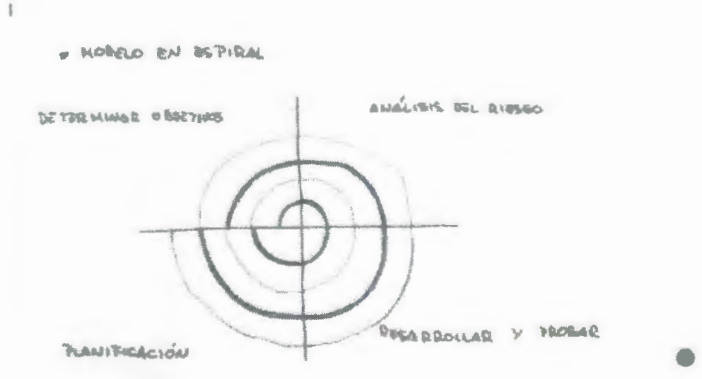


Figura 1-2 El modelo de desarrollo en espiral

La ventaja principal de este modelo se refiere al análisis del riesgo que se hace de forma explícita y clara. Une los mejores elementos de los restantes modelos.

1.2.1.3. Metodologías ágiles

Grupos de desarrollo han experimentado soluciones que basan su fundamento en la adaptabilidad de los procesos de desarrollo, en lugar de seguir esperando lograr resultados predecibles de un proceso que no evoluciona. Esta comunidad de desarrolladores e investigadores han nombrado su trabajo bajo lo que conocemos como metodologías ágiles. Las metodologías ágiles como puede entenderse mal, no están en contra de administrar procesos de desarrollo. Por el contrario promueve la formalización de procesos adaptables. Como característica fundamental, la habilidad de responder al cambio es la principal característica de las metodologías ágiles.

1.2.1.4. Metodologías de desarrollo iterativo

Es generalmente conocido que la tasa de éxito de los proyectos de desarrollo de software es notablemente más baja que la de cualquier otro proyecto de ingeniería. Los proyectos de software normalmente duran más de lo previsto, consumen más recursos y dinero de lo presupuestado y frecuentemente producen sistemas defectuosos, con una arquitectura rígida o inestable y con numerosos errores, muchos de los cuales sólo se detectan en tiempo de explotación, para un estudio cuantitativo del tema, ver **[7]**.

1.2.1.5. Proyectos con modelo de desarrollo iterativo

Como se puede deducir de lo dicho hasta ahora, un proyecto de desarrollo requiere un cambio constante. El modelo de desarrollo en cascada intenta evitar el cambio, fijando de forma temprana los requerimientos del sistema. En cambio, los modelos de desarrollo iterativos intentan adaptarse a este cambio, de ahí su idoneidad para el desarrollo de programas. Mi experiencia en el

asesoramiento y coaching a empresas indica que el uso de estas prácticas mejora sustancialmente el desempeño de los equipos, y hasta he detectado razones culturales favorables, que hacen que esta práctica sea "bien recibida" por los profesionales de estas latitudes.

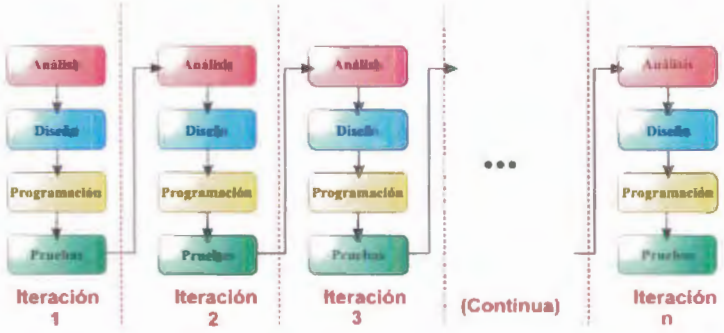


Figura 1-3 Un modelo de desarrollo iterativo

El esquema de un modelo iterativo se muestra gráficamente en la figura 1-3.

En esta tesis sólo hemos examinado la superficie de los modelos iterativos. Un estudio más profundo revelaría una serie de aspectos de suma importancia que no han podido incluirse aquí. Sin embargo, a pesar de la limitación del espacio, se confía en haber proporcionado al lector una idea general de qué tipo de modelo de

desarrollo es el más conveniente para su empresa y fomentado la inquietud de seguir informándose con más amplitud sobre ello.

1.3. Problemas de las metodologías de desarrollo actuales

1.3.1. Las promesas del desarrollo ágil han fallado

Esta fue una de las afirmaciones de Steve McConell **[9]**, que defendió su postura ecléctica sobre los modelos de desarrollo de software en su intervención "10 Most Important Ideas in Software Development" en el congreso SD WEST2006 (Software Development Conference & Expo).

1.3.2. Debilidades de las metodologías de desarrollo de software actuales

1.3.2.1. Proyectos sin modelo de desarrollo

Comencemos con los proyectos que no utilizan ningún modelo de desarrollo. Aunque la necesidad de un modelo de desarrollo hace décadas que está firmemente establecida, es triste comprobar

cómo, en nuestro entorno, la desidia y falta de profesionalidad así como un concepto corto de miras de la gestión empresarial hacen que la mayoría de proyectos de software no apliquen ningún modelo en absoluto. La filosofía subyacente a dichos proyectos suele ser que el análisis y diseño del sistema, así como cualquier planificación de su desarrollo, son una pérdida de tiempo y que lo importante es comenzar a programar cuanto antes para entregar el producto lo más pronto posible.

1.3.2.2. Debilidades de la metodología en cascada

Los programas informáticos son mucho más complejos y abstractos que un edificio o que cualquier producto resultante de otra ingeniería. Es por esto que este modelo no se adecua bien al desarrollo de sistemas. Sus principales defectos son los siguientes:

1. Rigidez y poca adaptabilidad. Un estudio realizado (**[10]**) revela que los requerimientos no anticipados en el comienzo del proyecto pueden suponer un 25% del total para proyectos de

desarrollo medios y hasta un 50% para proyectos grandes (similares resultados se presentan en el artículo **[11]**).

2. Baja mitigación de riesgos.

3. Falta de retroalimentación.

Como consecuencia, el modelo de desarrollo en cascada es demasiado rígido para un proceso tan dinámico como es el desarrollo de software. Es por eso que adoptarlo es contraproducente para la correcta ejecución del proyecto de software. De hecho, en un estudio de dos años sobre proyectos de desarrollo exitosos que se publicó en **[4],** se determinó que el primer factor de éxito para un proyecto de desarrollo es adoptar un modelo de desarrollo diferente del modelo en cascada.

1.3.2.3. Debilidades del modelo de desarrollo iterativo

Entre las principales debilidades de este modelo se mencionan las siguientes:

- Debido a la interacción con los usuarios finales, cuando sea necesaria la retroalimentación hacia el grupo de desarrollo, utilizar este modelo de desarrollo puede llevar a avances extremadamente lentos.

- Por la misma razón no es una aplicación ideal para desarrollos en los que de antemano se sabe que serán grandes en el consumo de recursos y largos en el tiempo.

- Al requerir constantemente la ayuda de los usuarios finales, se agrega un costo extra a la compañía, pues mientras estos usuarios evalúan el software dejan de ser directamente productivos para la compañía.

1.3.2.4. Encuesta sobre la aplicación de metodologías de desarrollo de software

Esta sección del documento recopila las preguntas, opiniones y respuestas que se produjeron en un pequeño curso sobre las metodologías de desarrollo de software, y en especial sobre metodologías ágiles recibidos en la Superintendencia de Bancos y Seguros. La cualidad de hacer debates vía correo electrónico ha facilitado la recopilación que se ofrece a continuación. Las preguntas y opiniones expresan las ideas que se tienen acerca de los temas tratados. Se han copiado textualmente de los documentos originales, salvo muy pocas excepciones donde se han realizado pequeños retoques para mejorar la claridad.

a) Aproximación convencional (Modelo en cascada)

Es una técnica rígida para mejorar la calidad y reducir los costos del desarrollo de software. Tradicionalmente se conoce como "modelo en cascada", porque su filosofía es completar un paso con

un alto grado de exactitud, antes de empezar el siguiente. Los principales problemas que se han detectado en esta aproximación son debidos a que se comienza estableciendo todos los requisitos del sistema. Una especificación del detalle de los posibles escenarios que pueden darse se presentan a continuación:

1. En muchas ocasiones no es posible disponer de unas especificaciones correctas desde el primer momento, porque puede ser difícil para el usuario establecer al inicio todos los requisitos. **R:** De acuerdo

2. En otras hay cambio de parecer de los usuarios sobre las necesidades reales cuando ya se ha comenzado el proyecto, siendo probables que los verdaderos requisitos no se reflejen en el producto final. **R:** De acuerdo

3. Otro de los problemas de esta aproximación es que los resultados no se ven hasta muy avanzado el proyecto, por lo

tanto la realización de modificaciones, si ha habido un error, es muy costosa. **R:** De acuerdo

Entre los problemas que se pueden encontrar con este modelo, se tienen:

1. Los proyectos raras veces siguen el modelo secuencial que se supone. Los cambios pueden causar confusión. **R:** *De acuerdo*

2. Es difícil disponer en principio de todos los requisitos. Este modelo presenta dificultades en el momento de acomodar estas incertidumbres. **R:** *De acuerdo*

3. La versión operativa de los programas no está disponible hasta que el proyecto está muy avanzado. Un error importante puede ser desastroso, si se descubre al final del proceso. **R:** *De acuerdo*

4. Los responsables del desarrollo siempre se retrasan innecesariamente. *Aunque puede suceder de forma accidental.*

   R: *No de acuerdo*

5. Algunos integrantes del equipo de desarrollo han de esperar a otros para completar tareas pendientes. (secuencial)**R:** *De acuerdo*

b) Aproximación prototipo

Es habitual que en un proyecto de desarrollo de software no se identifiquen los requisitos detallados de entrada, procesamiento o salida. En otros casos no se está seguro de la eficiencia de un algoritmo, o de la forma en que se ha de implantar la interfaz hombre-máquina. En casos así, lo habitual es construir un prototipo, que idealmente sirviera como mecanismo para identificar los requisitos del software. Las premisas clave de esta aproximación son:

1. Que los prototipos constituyen un medio mejor de comunicación que los modelos en papel.

2. Que la iteración es necesaria para canalizar, en la dirección correcta, el proceso de aprendizaje.

El problema, es que los usuarios finales, ven lo que parece ser una versión de trabajo del software, sin considerar que no es la versión definitiva y por lo tanto no se han considerado aspectos de calidad o facilidad de mantenimiento.

Cuando se les dice que el producto es, a partir de entonces, cuando se debe de empezar a "fabricar", no lo entiende y empieza de nuevo con ajustes, lo cual hace este proceso muy lento.

R: *Los prototipos pueden ser desechables o evolutivos. El primero es propio de la cascada, mientras que el segundo es propio de los*

desarrollos cíclicos. Los problemas citados pertenecen a los desechables, pero no a los evolutivos. El conflicto entre el concepto de prototipo y calidad es artificial.

c) Aproximación evolutiva

En esta aproximación el énfasis está en lograr un sistema flexible y que se pueda expandir de forma que se pueda realizar muy rápidamente una versión modificada del sistema cuando los requisitos cambien. Se diferencia de la aproximación anterior, en que en ésta los requisitos cambian continuamente, lo cual implicaría en el caso previo que las iteraciones no tendrían fin.

R: *Esta aproximación facilita enfrentar la incertidumbre cuando las cosas cambian. Si la diferencia entre esta aproximación y la anterior es la infinitud de las iteraciones, entonces la anterior se refiere a prototipos desechables.*

d) Aproximación espiral

Nace con el objetivo de captar lo mejor de la aproximación convencional y de la de prototipo, añadiendo un nuevo componente, el análisis de riesgos.

Esquemáticamente se puede ilustrar mediante una espiral, con cuatro cuadrantes que definen actividades.

En la primera vuelta de la espiral se definen los objetivos, las alternativas y las restricciones y se analizan y se identifican los riesgos. Si como consecuencia del análisis de riesgo se observa que hay incertidumbre sobre el problema entonces en la actividad correspondiente a la ingeniería se aplicará la aproximación prototipo cuyo beneficio principal es el de reducir la incertidumbre de la naturaleza del problema de información y los requerimientos que los usuarios establecen para la solución a ese problema (mucho mejor para eliminar incertidumbres que las anteriores, e

incorpora también determinación de alternativas, eliminación de ambigüedad).

Al final de esta primera vuelta alrededor de la espiral el usuario evalúa los productos obtenidos y puede sugerir modificaciones. Se comenzaría avanzando alrededor del camino de la espiral realizando las cuatro actividades indicadas a continuación. En cada vuelta de la espiral, la actividad de ingeniería se desarrolla mediante la aproximación convencional o ciclo de desarrollo en cascada o mediante la aproximación de prototipos.

- Actividades.
- Acciones.
- Planificación, determinación de alternativas, identificación y resolución de riesgos.
- Ingeniería.
- Desarrollo y verificación del producto de siguiente nivel.
- Evaluación del cliente.
- Valoración de los resultados del proceso de desarrollo.

R: *No elimina incertidumbre. La espiral está concebida como un camino prudente de enfrentar riesgos, sólo eso. Parece que la espiral realiza cada iteración según una cascada, y así se lo dijimos como crítica personalmente a Boehm, su autor. Pero él respondió que la espiral no contenía cascadas. Y nos alegramos, porque una iteración NO puede ser una cascada porque el concepto de iteración contradice al concepto de cascada.*

1.3.2.4.1. Respuestas a preguntas de los profesores y/o instructores

1. ¿Cómo medimos la incertidumbre? ¿Con métodos de calidad?

R: La incertidumbre se puede medir en términos de la cantidad de información que se necesita para resolver la incertidumbre. Pero, no creemos que sea necesario medirla en la construcción de software. ¿Qué son métodos de calidad? ¿Qué es calidad? ¿Hay una definición única?

2. ¿Cómo valoramos a priori el riesgo de la incertidumbre sobre un proyecto? ¿por el daño que puedan causar sus efectos nocivos?

R: *Riesgo es una terna formada por: un suceso, una consecuencia desfavorable, incertidumbre). Si falta alguno de sus componentes no hay riesgo. La valoración del riesgo integra a todos sus componentes.*

3. Con los métodos de prevención de riesgos, técnicas de seguridad, etc., evitamos incertidumbre, pero ¿cómo buscar el equilibrio de coste–beneficio entre los recursos asignados a un proyecto de más para evitar incertidumbre y los costes de no hacerlo?

R: *Los métodos citados NO evitan la incertidumbre. Si evitaran la incertidumbre eliminarían los riesgos y no lo hacen. La finalidad de tales métodos es tomar un seguro contra riesgos, que como seguro al fin, encarece el producto. No hay equilibrio, simplemente se trata de apostar, igual que en un seguro de cualquier otro tipo.*

4. ¿Es cierto que al final los métodos ágiles y las técnicas de desarrollo orientadas a objetos se adaptan mejor o manejan mejor la incertidumbre? Si esto es así por qué no se adoptan ¿por el coste? ¿por el nivel de desarrollo?

R: *Sí, los métodos ágiles y las técnicas orientadas a objetos facilitan mejor el manejo de la incertidumbre. Pero, como nada es gratis son más costosas. Por tanto, no siempre se justifican. Y además, su adopción general, como un camino más, requiere de un cambio cultural que se está produciendo, pero que no se ha completado aún.*

CAPÍTULO II

"DESCRIPCIÓN DE PROBLEMAS"

CAPÍTULO II

2. DESCRIPCIÓN DE PROBLEMAS

2.1. Situación actual

La Superintendencia de Bancos y Seguros (SBS) cuenta con una estructura orgánica de procesos por cada Dirección administrativa, a partir de esta estructura de procesos se forman los planes estratégicos y los planes de labores rutinarias en la Dirección Nacional de Recursos Tecnológicos (DNRT). En la siguiente parte del documento se presenta el detalle de la estructura orgánica de procesos utilizada en la DNRT.

2.1.1. Definición de procesos internos de desarrollo de aplicaciones

Existen dos tipos de estructuras que son las principales en la SBS:

1. Orgánica de unidades o direcciones; y,
2. Orgánica de procesos/subprocesos.

La parte de la estructura orgánica de unidades o direcciones de la SBS que afecta directamente al desarrollo de aplicaciones tecnológicas es la que se refiere a las siguientes direcciones o unidades administrativas de la SBS:

a) Dirección Nacional de Recursos Tecnológicos (DNRT); y,
b) Dirección Nacional de Desarrollo Institucional (DNDI).

La DNRT está conformada de dos subdirecciones:

1. Subdirección de desarrollo y aplicaciones tecnológicas (SDAT); y,
2. Subdirección de recursos tecnológicos (SRT).

La DNDI está conformada por dos subdirecciones:

1. Subdirección de Desarrollo Institucional (SDI); y,
2. Subdirección de Gestión de Recursos Humanos y Remuneraciones (SGRH).

Para cada Dirección administrativa de la SBS, se tiene definida una jerarquía de procesos definida, partiendo desde el nivel superior que se le denomina **macroproceso** llegando al nivel inferior de la jerarquía, al que se le denomina **proceso/subproceso.**

Los macroprocesos, (s)/subproceso(s) de todas las unidades o direcciones de la SBS se definen en la Subdirección de Desarrollo Institucional (SDI) que antes se denominaba Subdirección de Estrategia y Gestión (SEG) y que forma parte de la Dirección Nacional de Desarrollo Institucional (DNDI) que antes se denominaba Dirección Nacional de Estrategia y Gestión (DNEG).

Para el estudio de la metodología de desarrollo propuesta nos enfocaremos en un proceso/subproceso específico denominado **"Desarrollo, Implantación y Mantenimiento de Aplicaciones Tecnológicas"**, el cual pertenece a la SDAT. En la figura 2-1 se visualiza la estructura jerárquica de procesos de la DNRT, especificando además uno de los proceso(s)/subproceso(s) de la SDAT.

Figura 2-1 Estructura orgánica de procesos de la DNRT – SDAT

A continuación se presenta una parte del documento de políticas de todos los procesos de la SBS, en donde se indican las funciones y responsabilidades de la DNRT.

2.1.1.1. Las responsabilidades de la Dirección Nacional de Recursos Tecnológicos

Artículo 100.- La Dirección Nacional de Recursos Tecnológicos tendrá las siguientes funciones:

a) Presentar a la Dirección Nacional de Desarrollo Institucional y Recursos Humanos el plan operativo anual sobre la base de la estrategia institucional, proceder a su ejecución y generar informes semestrales sobre su grado de cumplimiento;

b) Presentar al Superintendente de Bancos y Seguros el plan estratégico tecnológico, que deberá considerar una infraestructura que garantice la continuidad, vigencia y agilidad de los servicios de manera segura e integral para soportar los procesos institucionales;

c) Proveer de las herramientas informáticas necesarias para la recepción de información que en medios electrónicos remitan las entidades controladas;

d) Proveer de las herramientas informáticas necesarias para la ejecución de los procesos institucionales, poniendo principal énfasis en los correspondientes a la Cadena de Valor, y presentar la documentación técnica así como los manuales de usuario y tutoriales de forma consolidada en los medios tecnológicos de difusión disponibles;

e) Elaborar las políticas, procedimientos y metodologías para proveer de asistencia técnica, a nivel nacional, a los usuarios cuando y donde éstos lo requieran, como soporte a la ejecución de los procesos institucionales;

f) Elaborar las políticas, procedimientos y metodologías para proveer de recursos tecnológicos, a nivel nacional, como soporte a la ejecución de los procesos institucionales;

g) Mantener actualizado el Mapa Tecnológico Institucional que debe estar alineado a los procesos institucionales y al Mapa de Información Financiera;

h) Aprobar el desarrollo de las estructuras de información de Balances, Patrimonio Técnico, Central de Riesgos, Mercado y Liquidez, Catastro, Central de Siniestros y Deudores en Mora, Cuentas Corrientes Cerradas y Cheques Protestados y de cualquiera otra información electrónica que deba ser enviada por las entidades controladas a la Superintendencia de Bancos y Seguros;

i) Aprobar el desarrollo de los validadores de la información de las estructuras a efectos de garantizar su confiabilidad y veracidad;

j) Elaborar el inventario nacional actualizado de los recursos tecnológicos y presentarlo a la Dirección Nacional de Finanzas y Recursos Materiales para su incorporación en el inventario de bienes institucionales;

k) Asignar los recursos tecnológicos institucionales, a nivel nacional, a los servidores de la Superintendencia, de acuerdo a las disponibilidades económicas, y llevar el registro correspondiente;

l) Asesorar al Superintendente de Bancos y Seguros y al Intendente General en asuntos de su competencia; y,

m) Las demás que le sean asignadas por el Superintendente de Bancos y Seguros o el Intendente General.

Artículo 101.- Para el cumplimiento de las funciones establecidas en el artículo anterior, la Dirección Nacional de Recursos Tecnológicos ejecutará los siguientes procesos y subprocesos:

Macro-proceso:

- Recursos Tecnológicos.

Procesos:

- Desarrollo, Implantación y Mantenimiento de Aplicaciones Tecnológicas.
- Administración de Recursos Tecnológicos.

Subprocesos:

- Operación de Recursos Tecnológicos.
- Soporte a Usuarios.

Para fines de la ejecución de los procesos y subprocesos bajo su responsabilidad, considerará que los resultados deben atender a los siguientes clientes internos:

- Superintendente de Bancos y Seguros, Intendente General, Comité Estratégico.
- Organizacional y las unidades administrativas que conforman la Institución.

Artículo 102.- La Dirección Nacional de Recursos Tecnológicos tendrá la siguiente estructura:

A) DIRECCION NACIONAL DE RECURSOS TECNOLOGICOS

a.1) Subdirección de Desarrollo y Aplicaciones Tecnológicas

a.2) Subdirección de Recursos Tecnológicos

2.1.1.2. Las responsabilidades de la Subdirección de desarrollo y aplicaciones tecnológicas

Artículo 103.- La Subdirección de Desarrollo y Aplicaciones Tecnológicas tendrá las siguientes funciones:

a) Presentar al Director Nacional de Recursos Tecnológicos el plan operativo anual sobre la base de la estrategia institucional, proceder a su ejecución y generar informes semestrales sobre su grado de cumplimiento;

b) Ejecutar el Plan Estratégico Tecnológico en el ámbito de su competencia;

c) Desarrollar las estructuras de información de: Balances, Patrimonio Técnico, Central de Riesgos, Mercado y Liquidez, Catastro, Central de Siniestros y Deudores en Mora, Cuentas Corrientes Cerradas y Cheques Protestados y de cualquiera otra información electrónica que deba ser enviada por las entidades controladas a la Superintendencia de Bancos y Seguros, una vez que sean aprobadas por la Dirección Nacional de Estudios;

d) Desarrollar o gestionar la adquisición de herramientas informáticas necesarias para la ejecución de los procesos institucionales, poniendo principal énfasis en los correspondientes a la Cadena de Valor; preparar la documentación técnica y los manuales de usuario y tutoriales de forma consolidada en los medios tecnológicos de difusión disponibles, así como los informes y estadísticas sobre el funcionamiento de las herramientas informáticas; y, proponer planes de mejoramiento;

e) Presentar propuestas de mejoramiento y estadísticas para la actualización permanente del Mapa Tecnológico Institucional alineado a los procesos institucionales y al Mapa de Información;

f) Desarrollar los validadores de la información de las estructuras a efectos de garantizar su confiabilidad y veracidad. La definición y aprobación de los validadores será de responsabilidad de la Dirección Nacional de Estudios;

g) Depositar en el buzón de los Burós de Información Crediticia, de manera oportuna, los archivos contentivos de la información que deba ser entregada de conformidad con los convenios suscritos para el efecto, debidamente aprobada por la Dirección Nacional de Estudios;

h) Asesorar al Director Nacional de Recursos Tecnológicos en asuntos de su competencia; y,

i) Las demás que le asigne el Superintendente de Bancos y Seguros o el Director Nacional de Recursos Tecnológicos.

Artículo 104.- Para el cumplimiento de las funciones establecidas en el artículo anterior, la Subdirección de Desarrollo y Aplicaciones Tecnológicas ejecutará el siguiente proceso:

Macro-proceso:

- Recursos Tecnológicos.

Proceso:

- Desarrollo, Implantación y Mantenimiento de Aplicaciones Tecnológicas.

Para fines de la ejecución del proceso bajo su responsabilidad, considerará que los resultados deben atender a los siguientes clientes internos: Superintendente de Bancos y Seguros, Dirección Nacional de Recursos Tecnológicos, Dirección Nacional de Finanzas y Recursos Materiales e Intendencias Regionales.

2.1.2. Planificación de proyectos de desarrollo de aplicaciones

En esta sección del documento se especifica los lineamientos a los cuáles se sujetan las tareas de desarrollo y de mantenimiento de aplicaciones tecnológicas propuestas por la SDAT.

Para planificar un nuevo desarrollo se debe tomar en cuenta la siguiente definición dada por el Director Nacional de Recursos Tecnológicos:

"La Planificación de desarrollo de aplicaciones para la SDAT tiene como finalidad proporcionar un marco de trabajo que permita hacer estimaciones razonables de recursos, costos y planificación temporal".

Siguiendo la misma línea en cada proceso de desarrollo o de mantenimiento de una aplicación tecnológica se deben cumplir las actividades o tareas definidas en el proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas".

El proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas" tiene como objeto específico permitir

que los usuarios internos y externos de la Superintendencia de Bancos y Seguros puedan contar con herramientas tecnológicas que faciliten la realización de sus tareas de manera eficiente.

Antes de realizar un nuevo desarrollo o mantenimiento de una aplicación tecnológica se deben analizar los siguientes puntos:

- Revisión del alcance de las tareas del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas".

- Revisión del manual de políticas del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas".

- Realizar la identificación de cuáles serán los clientes internos y externos de la nueva aplicación tecnológica, o de los clientes

que se relacionan con el mantenimiento de una aplicación tecnológica de la SBS.

En la actualidad la planificación de los desarrollos de proyectos de software no cumple a cabalidad las tareas o actividades definidas para el proceso/subproceso de desarrollo y mantenimiento de aplicaciones tecnológicas.

No forma parte de esta tesis la formulación de un nuevo proceso/subproceso de desarrollo, ni la definición a mejoras del proceso/subproceso existente, ya que el objetivo principal de esta nueva metodología de desarrollo de aplicaciones tecnológicas es establecer guías para una construcción ordenada de las aplicaciones. Por otro lado la actividad de definición de mejoras del proceso/subproceso "desarrollo y mantenimiento de aplicaciones tecnológicas" le corresponde a la Subdirección de Desarrollo Institucional de la SBS (la SDI antes Dirección Nacional de Estrategia y Gestión), en donde la Subdirección de Desarrollo y

Aplicaciones Tecnológicas sirve como soporte al mejoramiento de este proceso/subproceso.

El proceso/subproceso "desarrollo y mantenimiento de aplicaciones tecnológicas" afecta en forma directa al funcionamiento de la SDAT, pero el documento de éste proceso siempre está cambiando, por lo que en la mayoría de los casos la SDAT omite las recomendaciones del proceso y esto genera graves problemas, sobre todo en la descoordinación de las dos áreas (SDAT y SDI).

2.1.2.1. Alcance del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas"

Inicia desde el planteamiento de los requerimientos por parte de las diferentes unidades administrativas hasta la implantación de las aplicaciones con la respectiva capacitación a los usuarios y el mantenimiento de las aplicaciones una vez que han entrado a producción.

2.1.2.2. Políticas del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas"

Las prioridades y selección de proyectos informáticos deberán ser definidas por el Comité Estratégico Institucional en función de la estrategia institucional.

Las especificaciones técnicas del aplicativo deben estar acordes a las necesidades del proceso establecido por la Dirección Nacional de Desarrollo Institucional y Recursos Humanos (DNDI). Los usuarios del aplicativo a desarrollarse deberán participar en la definición de las especificaciones técnicas y la validación del modelo en la fase piloto.

2.1.2.3. Clientes internos y externos de las aplicaciones tecnológicas

Los clientes internos son los siguientes:

- Superintendente(a) de Bancos y Seguros.
- Dirección Nacional de Recursos Tecnológicos.
- Dirección Nacional Financiera Administrativa.
- Dirección Nacional de Desarrollo Institucional y Recursos Humanos.
- Intendencias Regionales.
- Comité Estratégico Institucional.

Los clientes externos son los siguientes:

- Grupo de entidades financieras controladas.
- Grupo de entidades del seguro privado.
- Grupo de entidades del seguro social.
- Cooperativas de ahorro y crédito controladas.

2.2. Flujo del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas"

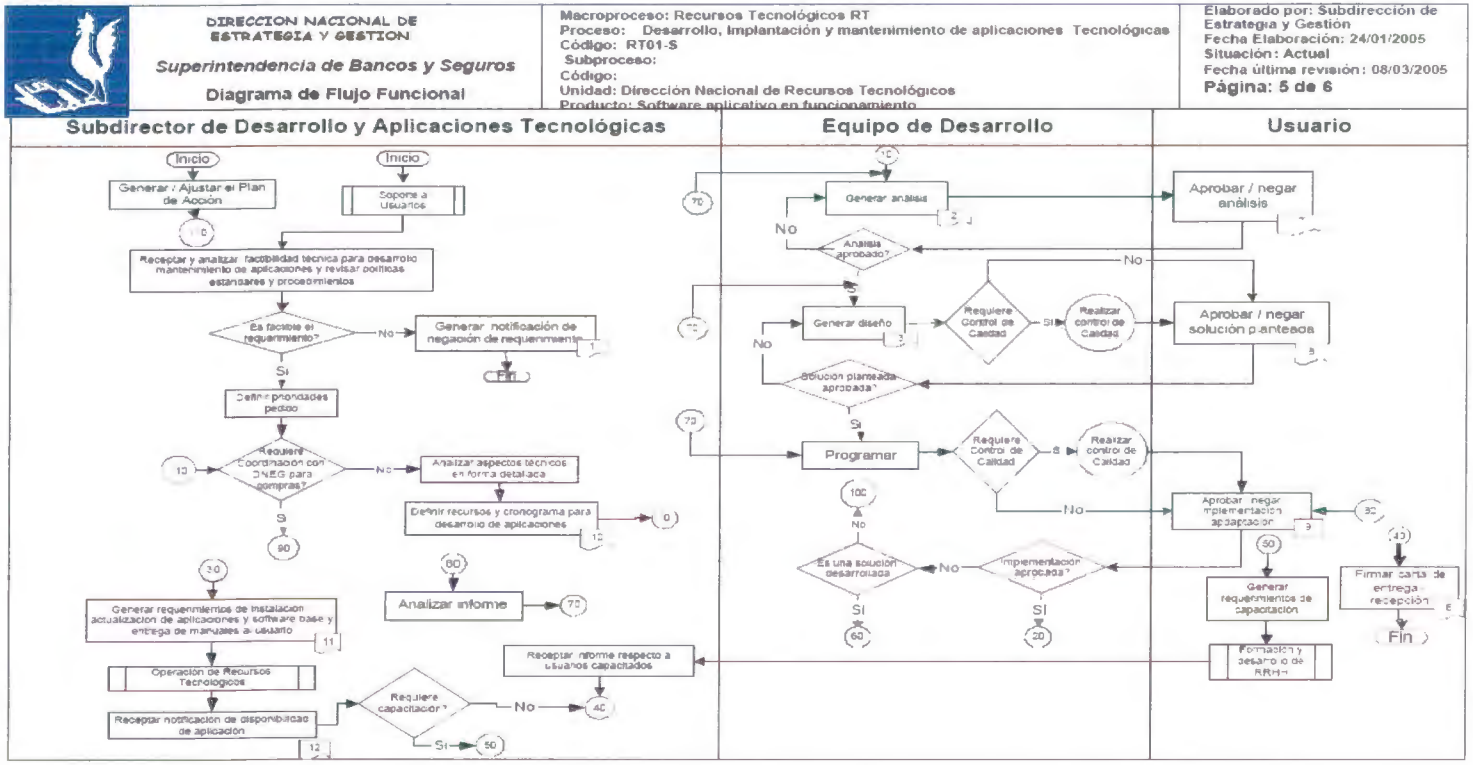


Figura 2-2 Flujo del proceso de desarrollo de aplicaciones (parte 1 de 2)

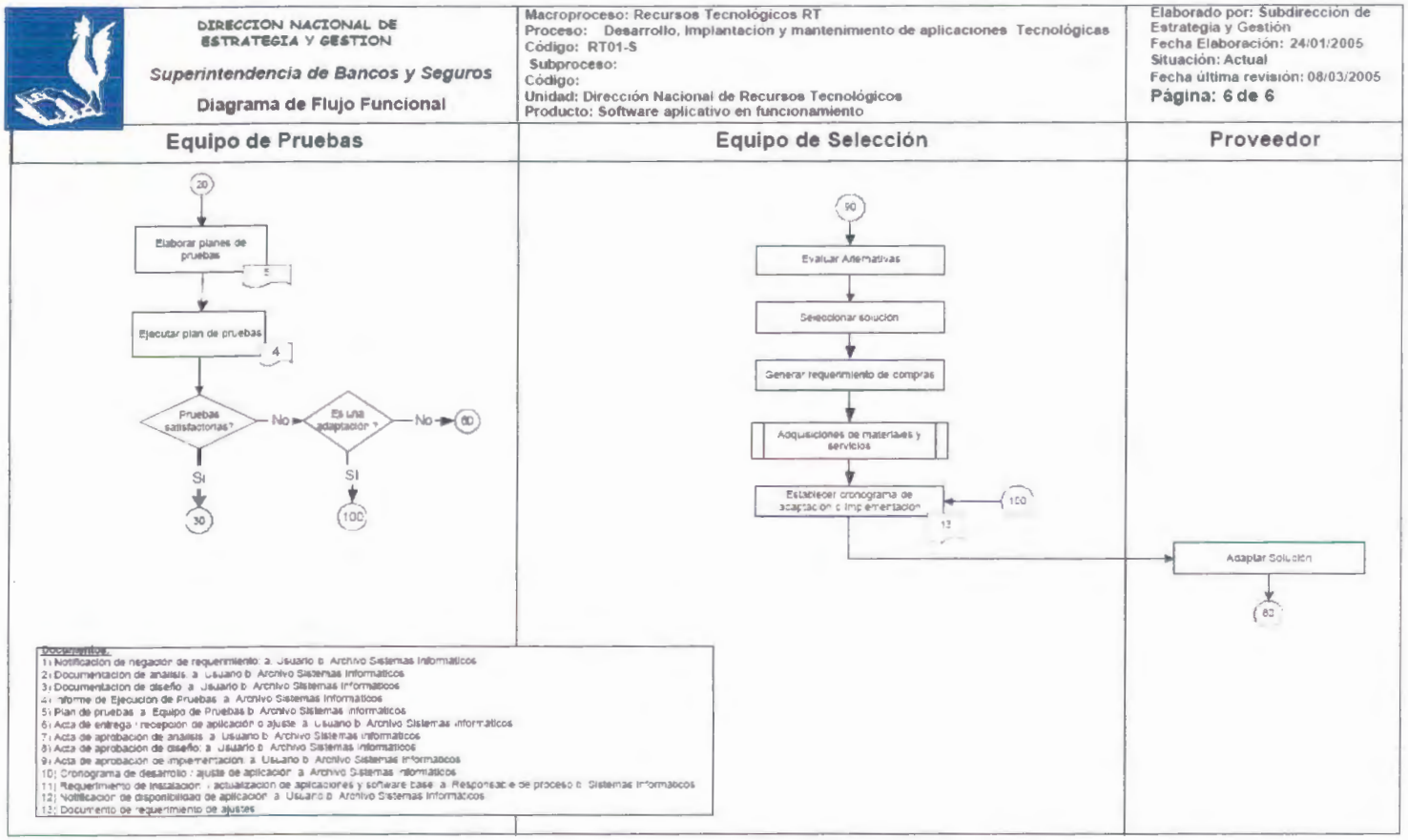


Figura 2-3 Flujo del proceso de desarrollo de aplicaciones (parte2 de 2)

2.2.1. Descripción del diagrama de flujo del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas"

El procedimiento inicia con el cumplimiento de los proyectos incluidos en el Plan Estratégico Informático Tecnológico, o en la atención de los requerimientos planteados a Soporte Técnico por los usuarios cuando el encargado del proceso ha elevado el nivel del soporte técnico (mantenimiento de aplicaciones).

El siguiente paso es el análisis de factibilidad técnica del requerimiento y la revisión de estándares, políticas y procedimientos para determinar si el requerimiento es viable (Política de solicitud, selección y definición de prioridades de proyectos informáticos); si no es viable se genera la notificación de negación del requerimiento. Si la viabilidad del proyecto determina la compra de un software o la tercerización de la programación, la DNEG para la evaluación y selección de alternativas y para la determinación de periodos de prueba según la oferta.

Luego se remite a la instancia respectiva para la adquisición del software o tercerización. En caso de ser una solución a desarrollarse se define prioridades y se establecen los recursos necesarios y se define el cronograma inicial de actividades.

Luego se pasa al análisis y estudio de requerimientos, en el cual participan los usuarios contraparte, tanto en la aprobación como en la generación del análisis, obteniendo el documento respectivo y el acta de aprobación correspondiente.

Una vez aprobado el documento de análisis, se procede a la etapa de diseño en la que se determina el modelo arquitectónico de los proyectos de acuerdo a los estándares existentes. En esta fase se genera la documentación de diseño que debe ser aprobada por los usuarios con la firma del acta de aprobación.

Una vez aprobada la solución planteada en el diseño, se procede al control de calidad y se continúa con la programación del proyecto, que consiste en traducir y plasmar en una base de datos el diseño arquitectónico del proyecto.

De esta etapa se obtiene el acta de aprobación por parte de los usuarios y la documentación de la aplicación, se realiza un control de calidad de la programación; en caso de no ser aprobada, se realizarán los ajustes necesarios, según los resultados obtenidos.

A continuación de la aprobación de la programación, se elaboran los planes de prueba con el equipo de pruebas y se procede a su ejecución. Si las pruebas no son satisfactorias el Subdirector de Desarrollo y Aplicaciones Tecnológicas analiza el Informe de ejecución del plan de pruebas y remite al Equipo de Desarrollo para que realice los ajustes necesarios hasta que se obtenga la aprobación de la ejecución del plan de pruebas.

Luego de la obtención satisfactoria de las pruebas se generan los requerimientos de instalación, actualización de aplicaciones y software base, que se transmiten al proceso de Operación de Recursos Tecnológicos, proceso del que se obtendrá la notificación de disponibilidad de la aplicación.

2.2.1.1. Indicadores aplicados en el proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas"

Eficacia: Incremento de la producción institucional.

Medición: Productos generados en los sistemas automatizados de procesos / total de productos generados en la Institución.

Efectividad: Incremento de satisfacción del cliente interno.

Medición: (Productos generados en los sistemas automatizados / Total de productos generados en la Institución) * 0.50 + (Encuestas de Internet) * 0.50.

2.2.1.2. Problemas encontrados en la aplicación del proceso/subproceso "Desarrollo, implantación y mantenimiento de aplicaciones tecnológicas"

En esta parte del documento se especifica una lista de problemas que se han presentado en el proceso/subproceso para el desarrollo, implantación y mantenimiento de aplicaciones tecnológicas.

Los líderes de proyectos designados por el Subdirector de la SDAT se enfrentan constantemente a la presión por acortar el ciclo de desarrollo e implementación de las aplicaciones. A pesar de que proporcionar software de calidad a tiempo y dentro del presupuesto no ha sido una tarea fácil, ahora resulta más complicado debido a la adopción de las nuevas arquitecturas de múltiples niveles por parte de la Dirección Nacional de Recursos Tecnológicas de la Superintendencia de Bancos y Seguros.

Los problemas actuales, que se enfrentan en el desarrollo de software son los mencionados a continuación:

- Entornos de desarrollo y de producción que no coinciden.

- Representación inadecuada que aumenta el riesgo de la implementación de aplicaciones nuevas.

- Problemas de comunicación entre el equipo de desarrollo de la SDAT con el equipo administrador de los equipos servidores y de los equipos de telecomunicaciones en la SRT.

- Las pruebas se realizan incompletas durante la realización del software, y se espera al final del desarrollo del sistema para realizar nuevas pruebas. Esto causa retrasos en la atención a los clientes internos y resulta ser una situación costosa devolverse al origen de los posibles errores, aumentando el esfuerzo en el desarrollo del software y no se puede brindar un software de calidad al cliente.

- Los problemas que se quieren resolver con una aplicación tecnológica no cubren todos los requerimientos de los clientes internos, por lo que se tienen que generar múltiples versiones de los programas y el mantenimiento de las aplicaciones se vuelve ineficiente.

- Generalmente toma meses de revisión exhaustiva para que unos cuantos alcancen la seguridad de que los errores han sido eliminados del todo.

- La documentación de las aplicaciones que se encuentran en el ambiente de producción y desarrollo, no tiene una adecuada administración debido a que no llega de forma oportuna para que los técnicos de la SDAT puedan dar un mantenimiento eficiente y eficaz sobre éstas aplicaciones.

- Los documentos y/o manuales técnicos de las aplicaciones tienen ciertas deficiencias cómo por ejemplo: no están completos, no se encuentran actualizados con la última versión de las aplicaciones a las cuales se les ha hecho mantenimiento, no tienen un formato estándar para su elaboración.

- Existe un cierto número de usuarios distribuidos en las áreas cliente de la Superintendencia de Bancos y Seguros que tienen un perfil tecnológico y este a su vez detecta errores diferentes en las aplicaciones debido a que tienen diferentes maneras de evaluar a los programas. Cada uno enfoca la tarea de la caracterización de los errores con instrumentos analíticos distintos, desde un ángulo diferente.

- La administración insuficiente de requerimientos.

- Los problemas que afectan la comunicación, es ambigua e imprecisa.

- Las inconsistencias no detectadas entre requerimientos, diseño y programación.

- Las validaciones tardías de requerimientos.

- El enfrentamiento reactivo de riesgos.

- La propagación de cambios sin control.

- Las necesidades y expectativas de los clientes y usuarios no son captadas satisfactoriamente.

- Poca implicación del usuario final en todo el proceso.

- No se toma en cuenta el utilizar una metodología de desarrollo, sobre todo no se piensa aplicar una cuando se trata de proyectos pequeños de dos o tres meses.

- Aplicaciones muy sensibles a los cambios o modificaciones (mantenimiento), originadas ya sea por factores internos (sugerencias de los clientes internos) o por factores externos (Modificaciones en leyes y/o regulaciones).

- Al analizar el problema, no se hacen las preguntas correctas a las personas correctas.

- Los usuarios o clientes externos cambiaron su forma de pensar o sus percepciones.

- No existe una documentación sistematizada de los cambios que se realizan sobre las aplicaciones.

2.3. Documentación utilizada en el proceso de desarrollo de aplicaciones

2.3.1. Hojas de instrucción

- Circulares de instrucción emitidas en la Superintendencia de Bancos y Seguros.
- Instructivo de relevamiento y análisis de requerimientos (caracterización de proyectos).
- Instructivo de diseño de proyectos (Por definir).
- Instructivo de elaboración de documentación de aplicaciones (Por definir).
- Instructivo de preparación y ejecución de pruebas de aplicaciones (Por definir).

2.3.2. Formatos y anexos

- Formato de caracterización de proyectos.
- Estándares de diseño de interfaz.
- Estándares de programación.
- Estándares de base de datos.
- Formato de solicitud de aplicaciones (Por definir).

2.3.3. Documentación de soporte

- Estándares de Análisis, Diseño y Programación.
- Estándares de Control de Calidad.

2.3.4. Problemas encontrados en el manejo de la documentación en el flujo del proceso

No existe un control de ejecución de procesos que permita administrar los procedimientos y documentos, incluyendo todos los componentes básicos relacionados con el flujo del trabajo.

Entonces existe un costo de oportunidad que se asume por el área de la DNRT al no ejecutar las siguientes tareas o actividades:

- Tareas y actividades que integran el procedimiento.

- Usuarios responsables de la ejecución de las actividades.

- Tiempos de respuesta esperados no son reales.

- La documentación asociada al proceso/subproceso (hojas de instrucción, formatos y anexos, documentos de soporte), siempre está desactualizada porque la SDAT no maneja una gestión de requisitos adecuada durante el proceso de desarrollo y mantenimiento.

- Manuales, reglamentos, instructivos también se encuentran desactualizados.

- La correspondencia de tecnología no está normalizada.

- No existen alarmas asociadas al incumplimiento de actividades y vencimiento de plazos en el desarrollo de un proyecto.

2.4. Estándares actuales de desarrollo

Existe una documentación predefinida con los lineamientos para el desarrollo y mantenimiento del software en la Subdirección de Desarrollo y Aplicaciones Tecnológicas (SDAT), pero que a pesar de ello, no existen pruebas de buenas prácticas y de resultados integrales.

La filosofía que en la mayoría de los casos predomina en los ingenieros de sistemas de la SDAT, es la de realizar su propia metodología de desarrollo, que al final es un problema, ya que existen varias metodologías empíricas de desarrollo de software, una por cada técnico y al final no existe una metodología de desarrollo que permita hacer una aproximación más exacta al trabajo estandarizado, que permita lograr un desarrollo de aplicaciones más eficiente y a la vez eficaz.

Además varias metodologías de desarrollo por persona involucra, que si no existe documentación de éstas, entonces las personas que cubran los espacios de las personas que manejaban su propia metodología de desarrollo, tendrán que necesariamente ver si comprenden la metodología usada anteriormente (de la cual no hay documentación) o si aplican su propia metodología.

Existe también un control de calidad que no llena las expectativas de un buen desarrollo de software. Es decir en muchas ocasiones no se lo realiza por supuestas premuras de salida a producción de un software, o los controles de calidad sobre un software desarrollado no se realizan a cabalidad. No obstante existen unos indicadores que ayudan a medir el control de calidad de un software, pero que finalmente no garantizan la calidad del desarrollo del software.

Finalmente uno de los estándares para el proceso de desarrollo de las aplicaciones es que por cada fase del proceso tiene que existir un documento aprobatorio firmado por el o los usuarios finales, para los

cuales se desarrolla la aplicación, y la mayoría de los casos no se firman éstos documentos, lo cual trae luego muchos problemas para la SDAT, al no tener respaldos de la aceptación de los desarrollos por cada fase del proceso. Esto se debe también en parte a la mala redacción de los documentos aprobatorios, y es por esta razón que los usuarios finales no quieren comprometer su consentimiento de aprobación al no estar totalmente de acuerdo.

2.4.1. Problemas encontrados en los estándares de desarrollo actuales de la SDAT

Como antecedente a los problemas de estándares de desarrollo en la SDAT, cabe mencionar que la unidad no dispone de un ambiente estable para el desarrollo y mantenimiento de software. Aunque se utilicen técnicas correctas de ingeniería, los esfuerzos se ven minados por falta de planificación. El éxito de los proyectos se basa en la mayoría de las veces en el esfuerzo personal, aunque a menudo se producen fracasos y casi siempre retrasos y sobrecostes. El resultado de los proyectos es impredecible.

En la SDAT no hay un control de la gestión de proyectos de software efectivo, porque si bien es cierto que la unidad dispone de procedimientos y técnicas formales proporcionadas por la Subdirección de Desarrollo Institucional de la SBS (SDI), tanto de gestión como del proceso, y de herramientas, éstas técnicas y procedimientos no se utilizan de manera estándar en todos los proyectos.

Cuando los mencionados documentos y herramientas asociados al proceso/subproceso de desarrollo y mantenimiento de software de la SDAT se utilizan, pues en la mayoría de los casos se tiene como resultado un archivo más de mala práctica en el uso de los estándares y se logra un sistema no estándar. El trabajar con sistemas no estándares le está costando mucho dinero a la SDAT y a la SBS. El no utilizar siempre los estándares para un proyecto de software, trae como resultado la lista de consecuencias que se describe a continuación:

1. Algunas herramientas tecnológicas de la SBS son exóticas, es decir, demasiado personalizadas a un técnico de codificación sin manuales técnicos claros, lo cual trae por consiguiente que existan pocos especialistas para dar mantenimiento al software respectivo. Debido al soporte limitado para conseguir alguien que pueda resolver un problema complicado, conlleva a costos elevados para la SBS.

2. Muchas veces es poco más que imposible hacer una migración a una nueva plataforma tecnológica. Es por esta razón que muchas de las aplicaciones tecnológicas están expuestas a problemas de seguridad así como a problemas de rendimiento. Inclusive podría ser difícil o imposible migrar de hardware, en caso de contingencia.

3. Por la naturaleza de este tipo de herramientas es posible que no puedan crecer al ritmo de las exigencias de la SBS. Muchas aplicaciones tecnológicas no pueden crecer ahora, y la única opción que maneja la SDAT es asignar parte de su

presupuesto al problema, en términos de hardware y software. Se debe tomar en cuenta que la escalabilidad de un problema en términos de hardware no es lineal. En la mayoría de soluciones al problema de escalabilidad de las aplicaciones se suele duplicar la capacidad de procesador, lo cual no necesariamente reduce a la mitad los tiempos de respuesta, aun paralelizándolo como lo explica la ley de Amdahl **[12]**.

4. Existe un proveedor para la aplicación tecnológica de administración del RRHH pero sólo como soporte en la SBS, y de este caso se origina un problema de "**vendor lock in**", es decir, que una vez que la aplicación tecnológica de este tipo se convirtió en una parte vital para el funcionamiento de la SBS, el proveedor cobra lo que él quiere sin que la SBS pueda hacer mucho para evitarlo.

La curva de aprendizaje de todas las aplicaciones tecnológicas de la SBS suele ser muy alta, costando muchas horas hombre obteniendo un costo-beneficio mucho menor del que se

esperaba, ya que son conocimientos poco reusables fuera del campo original del problema a tratarse. En la mayoría de los casos se trata de aplicaciones muy personalizadas sin ser absolutamente necesario.

CAPÍTULO III

"SOLUCIÓN PROPUESTA"

CAPÍTULO III

3. SOLUCIÓN PROPUESTA

3.1. Definición de la metodología de desarrollo de aplicaciones Web

3.1.1. Antecedentes

Toda empresa, hoy en día, necesita estar presente en Internet, transmitiendo de manera eficaz sus objetivos. Por esta razón las últimas tecnologías deben estar al alcance para brindar servicios de información en línea. Entonces nace una problemática que consiste en seguir una metodología adecuada para el desarrollo del nuevo servicio Web que se quiere dar en la empresa, sea éste un portal de servicios, aplicaciones Web, etc.

3.1.2. El método

En este capítulo nos centramos en los problemas y métodos aplicados al desarrollo de software mediante una investigación en una Ingeniería concreta: la Ingeniería del Software. Discutimos, por una parte, la naturaleza del método en este campo. Por otra parte se analiza, mediante un caso de estudio, la similitud de los métodos de investigación en Ingeniería del Software con respecto a los métodos de desarrollo de software. Cuando la creatividad marca el proceso de investigación, hablamos de métodos creativos. Estos métodos se basan en características como la imaginación, premonición, visualización y en ellos interviene la inteligencia creativa del investigador por encima de la racional. La tabla I, resume los principales objetos de estudio, ciencias y métodos utilizados en la disciplina de la IS.

	Ciencia	Objeto de Estudio	Carácter	Métodos
TIPO A	Ciencias de la Ingeniería del Software	Construcción de nuevos objetos	Ingenieril	Cualitativos Creativos
TIPO B	Ciencias del Software	Objeto construido	Empírico	Cuantitativos
TIPO C	Ciencias de los Sistemas de Información	Implantación y uso de Objetos construidos	Cultural y Social	Cualitativos

Tabla I. Resumen de problemas, ciencias y métodos de la IS

Por todo lo dicho, parece que un método de investigación para abordar problemas propios de las *Ciencias de la IS*, debería estar basado, fundamentalmente, en métodos cualitativos y creativos. Lógicamente, no existe un único método. En contra de la opinión de Bunge, pensamos que no existe un método universal de resolución de problemas, sino que cada problema requiere su propio método. Existen varios autores que coinciden con esta opinión **[13]**. Así, por ejemplo, Popper dice: *"Por regla general, empiezo mis clases sobre el Método Científico diciendo a mis alumnos que el método científico no existe"*. De hecho, en la definición de esta metodología, un método para, por ejemplo la definición de un modelo nuevo, consistirá fundamentalmente, en estudiar los modelos existentes, reflexionar acerca de ellos, determinando sus ventajas y limitaciones, y plantear un nuevo modelo que, manteniendo las ventajas de los modelos estudiados, supere, en la medida de lo posible, las limitaciones de los mismos.

3.1.3. Paralelismo entre el Método en IS y el Método en el Desarrollo Software

Para justificar esta similitud entre el método de investigación en Ciencias de la IS y el método de desarrollo software, utilizamos un caso de estudio. El caso se basa en el método de investigación que se está utilizando para la especificación de un método para el desarrollo de Sistemas de Información Web. Nótese que se trata de un problema de carácter ingenieril y que hemos clasificado dentro del tipo A; es en este tipo de investigación en el que nos centramos. El método de investigación, se basa en los pasos que, según Bunge, se deben seguir en toda investigación científica (ver figura 3-1).

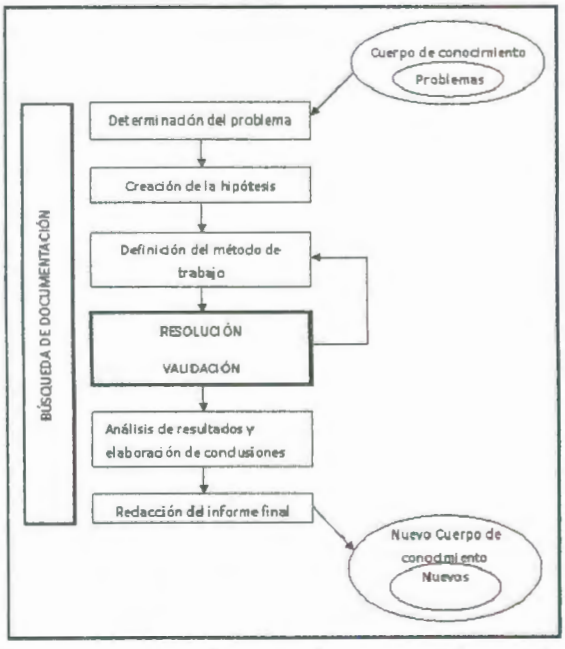


Figura 3-1 Método de Investigación

El centro de esta tesis se refiere a encontrar las principales similitudes con respecto a los métodos de desarrollo de software. La tabla II muestra un resumen de dichos paralelismos.

Etapas	Método de Investigación en Ciencias de la IS	Método de Desarrollo De Software
E1:Documentación	Problema a resolver Método de investigación	Dominio de la aplicación Metodología, técnicas, herramientas
E2:Determinación del Problema	Estudio de campo Delimitación del problema	Análisis de dominio Captura de requisitos
E3: Creación Hipótesis	Descripción del objeto a construir	Especificación de requisitos

Tabla II. Métodos de Investigación vs. Métodos de Desarrollo

Un método de desarrollo de software nos da las pautas para la construcción de nuevos objetos (software), al igual que el método de investigación nos da las pautas para la construcción de nuevos objetos (metodologías, modelos, etc.).

Si analizamos, por ejemplo, la filosofía de desarrollo propuesta por Beck en su metodología XP **[15]** (Extreme Programming), podemos encontrar que es muy cercana en cuanto a planteamientos a los

métodos basados en investigación en acción. En ambos casos se trata de obtener el producto (software, o metodología de desarrollo en nuestro caso de estudio), mediante la creación directa del mismo (programación o concepción y diseño), en colaboración con los usuarios (del software o de la metodología), probándolo y adaptándolo continuamente durante el propio proceso de creación. Al finalizar por ejemplo el desarrollo de un software, el conocimiento inicial del cliente a cerca de lo que puede obtener se habrá modificado (en general, se habrá ampliado) y los problemas de los que partimos ya no son los mismos. Algunos se habrán solucionado, otros se habrán modificado (reduciéndose o no) y aparecerán, además, nuevos problemas fruto de la implantación del software desarrollado. Este nuevo cuerpo de conocimiento con sus nuevos problemas dará lugar a otros trabajos de desarrollo.

3.1.4. Definición de la metodología de desarrollo

La metodología de desarrollo propuesta en este documento debe tomarse como una guía, pero no como algo rígido; debe adaptarse para cada utilización de la misma.

En las siguientes secciones se explica al detalle los aspectos internos y externos a la metodología de desarrollo, es decir el paradigma metodológico orientado a objetos y la metodología concreta la cual se trata de una recopilación de las mejores prácticas de las diferentes metodologías existentes, luego de un análisis de los resultados obtenidos con cada una de ellas.

3.2. Diseño de la arquitectura tecnológica

3.2.1. Arquitectura Tecnológica y el Proceso de Crear

Es importante hacer un análisis de las diferentes formas de crear que tenemos en la actualidad los profesionales en el campo tecnológico, sobre la base de conocimientos adquirida en los estudios académicos y de las experiencias obtenidas en el aspecto laboral.

3.2.1.1. Marco lógico

Una pregunta frecuente que se escucha de los ejecutivos de tecnologías de información es: ¿cuál es la arquitectura tecnológica adecuada para mi organización? creo que un buen punto de partida para analizar esto es un estudio que permita caracterizar adecuadamente los diferentes niveles de madurez de las organizaciones en esta área.

Hago entonces referencia a un estudio realizado por el Sloan Center for Information Systems Research del MIT en el año 2006, en el cual se pudo tipificar la evolución de las organizaciones respecto del estado de sus arquitecturas tecnológicas empresariales en cuatro niveles de madurez, lo cual se ve reflejado no sólo en las características de su arquitectura sino también en sus prácticas de gestión asociadas. Este es un proceso por el que deben transitar todas las organizaciones.

a) **Nivel I: Silos (Departamentales)**: Este es el nivel en el cual se puede afirmar que se encuentran las TICs, al referirse a la migración tecnológica que se realiza de forma progresiva y algo desordenada en la Superintendencia de Bancos y Seguros.

b) **Nivel II: Estandarización**: Este trabajo fue realizado para aplicar estándares de codificación y de documentación en el desarrollo de aplicaciones Web, tomando en cuenta las nuevas tecnologías que serán utilizadas en el proceso de migración de tecnologías antiguas, en la Superintendencia de Bancos y Seguros.

c) **Nivel III: Optimización del Núcleo**: Con la aplicación de la nueva metodología de desarrollo se espera que la SBS se llegue a enmarcar dentro de este nivel, con la finalidad de mejorar los servicios que ésta presta, y llegar a un nivel en cual los nuevos desarrollos puedan utilizar librerías o

componentes reusables para reducir esfuerzos, costo y tiempos en las planificaciones de tecnología.

d) Nivel IV: Modularización: A partir de las recomendaciones realizadas en esta tesis, la misma SBS realizó un análisis de la inversión en tecnologías muy sugerente, el cual refleja claramente donde se ponen los acentos.

3.2.1.2. Definición de Arquitectura de Software

La Arquitectura de Software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajemos en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. Existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura:

- La visión **estática**.
- La visión **funcional**.

- La visión **dinámica**.

Las arquitecturas más universales son:

- Monolítica.
- Cliente-servidor.
- Arquitectura de tres niveles.

3.2.1.3. Usabilidad y Arquitectura de Software

El Iceberg de la usabilidad es un concepto que se ha venido discutiendo desde los inicios de las aplicaciones Web. Se podría decir que, en el diseño de un sistema, hay tres aspectos a tener en cuenta:

- La presentación de la información.
- La funcionalidad de la aplicación.
- La Arquitectura del Software.

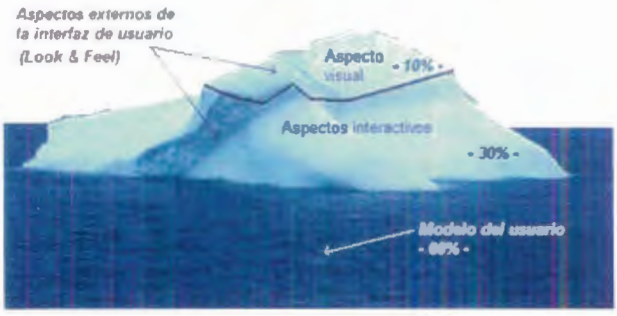


Figura 3-2 Iceberg de la usabilidad

Fuente: Ing. Dick Berry – integrante del IBM Ease of Use Team

Berry relaciona el modelo del usuario con el modelo de objetos propio de la interfaz de usuario, en el sentido estricto de la OOP (programación orientada a objetos), que incluye, entre otras cosas, los objetos y las metáforas de la interfaz. Este modelo de objetos, siempre según Berry, es el que permite al usuario relacionar sus objetivos con la funcionalidad del sistema.

"La tecnología es el motor que impulsa el diseño de interfaces. Esta sinergia es sin embargo una espada de dos filos: aún cuando el poder de la tecnología nos permite realizar grandes logros, nos ata de manera simultánea a maneras de pensar que son contrarias a la dirección natural del comportamiento humano. Casi todos los problemas con las interfaces-usuario modernas se originan en

diseñadores inteligentes y bien intencionados, pero que se concentran en los aspectos equivocados. En lugar de tecnologías y procedimientos, debemos dirigir nuestras energías a los objetivos fundamentales del usuario, aún cuando ellos mismos los desconozcan."

Alan Cooper
"Padre" de Visual Basic

3.2.1.4. La Arquitectura del Software propuesta para la metodología: "J2EE basado en el patrón de diseño MVC"

Esta propuesta considera el uso del *Framework ADF* para la capa de presentación mediante el uso de JSPs y JSFs, y para la capa de la lógica de negocio EJBs 3.0, de tal manera que permita a los objetos de negocio y de acceso a datos ser reusables, basados en lenguaje Java para utilizar los servicios específicos de J2EE y de entornos administrados de JDO.

Estos objetos de negocio persistentes pueden ser reutilizados en otros entornos J2EE (web o EJB), y en aplicaciones *standalone*, de

una manera sencilla. En otras palabras, los objetos persistentes pueden ser manipulados por la capa de presentación mediante la implementación de los métodos de los servicios de *ADF* que manipulan a modo *back-end* los objetos de negocio de la capa de integración, desacoplando así la capa de la lógica de negocio de la tecnología usada en la capa de integración. Para la capa de presentación, diversos *frameworks* de aplicaciones Web están disponibles, entre ellos: *Struts*, *WebWork* y *Java Server Faces* (JSF). De los cuales se ha considerado el uso de *Struts, y JSF,* el cual implementa el Modelo 2 del MVC (*Model-View-Controller*). Se presentan a continuación dos diagramas para ir explicando la distribución de los componentes en la arquitectura J2EE. El primer diagrama consiste en una vista lógica que muestra los componentes y servicios típicos de un entorno J2EE.

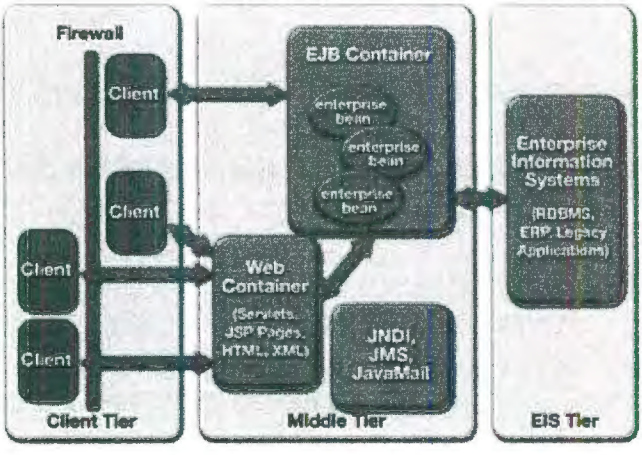


Figura 3-3 Diagrama lógico de componentes y servicios J2EE

El segundo diagrama es una vista de proceso que muestra las relaciones entre las capas model, view y controller de la arquitectura MVC bajo J2EE.

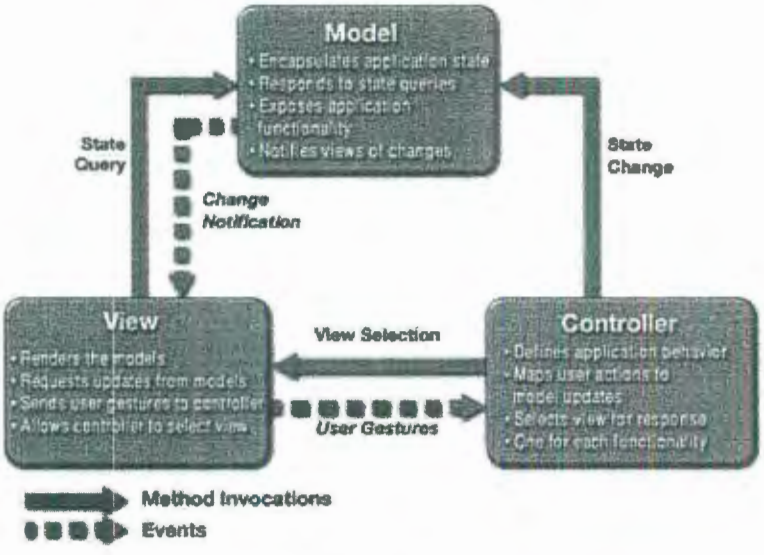


Figura 3-4 Relaciones entre las capas de la arquitectura MVC

3.2.1.5. Oracle Application Development Framework (ADF) para la administración técnica de la arquitectura de software

"Oracle Application Development Framework" (ADF) simplifica muchas de las tareas de desarrollo de aplicaciones J2EE haciendo innecesario la implementación de patrones o codificación de tareas

repetitivas. Oracle ADF implementa un conjunto de patrones de diseño que pueden ser reutilizados.

Oracle ADF tiene una fuerte acogida entre los desarrolladores Java gracias a que cuenta con las siguientes características:

1. Entorno de desarrollo Visual.
2. Plataforma Independiente.
3. Variedad de Opciones tecnológicas.
4. Solución de Extremo a Extremo.
5. Desarrollo Rápido.

Oracle ADF se basa en el patrón de diseño Modelo – Vista – Controlador (MVC) ver Figura 3-5.

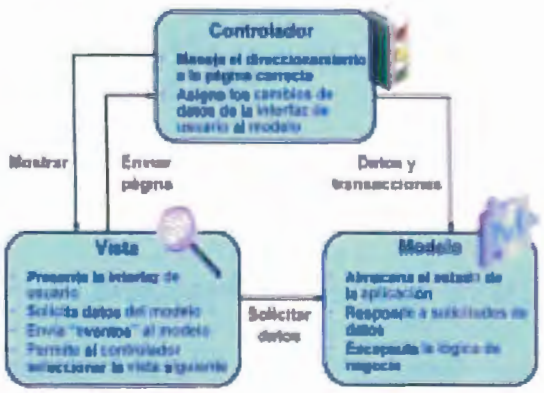


Figura 3-5 Patrón MVC

repetitivas. Oracle ADF implementa un conjunto de patrones de diseño que pueden ser reutilizados.

Oracle ADF tiene una fuerte acogida entre los desarrolladores Java gracias a que cuenta con las siguientes características:

1. Entorno de desarrollo Visual.
2. Plataforma Independiente.
3. Variedad de Opciones tecnológicas.
4. Solución de Extremo a Extremo.
5. Desarrollo Rápido.

Oracle ADF se basa en el patrón de diseño Modelo – Vista – Controlador (MVC) ver Figura 3-5.

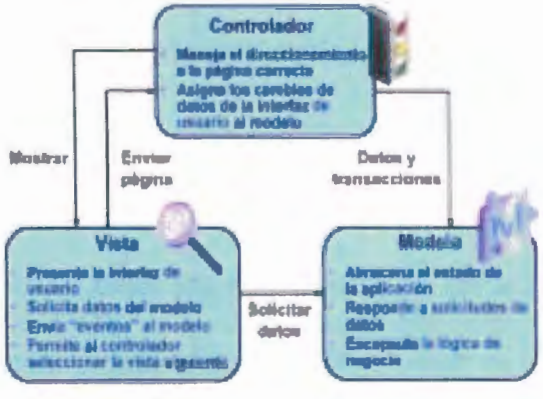


Figura 3-5 Patrón MVC

3.2.1.6. Definiendo el rol de "Arquitecto de software"

La presente sección trata sobre las posturas que, a veces, se ve obligado a tomar un arquitecto de software, relacionando al arquitecto y su arquitectura, con un visionario y su visión, respectivamente. Las categorías de arquitectos de software recomendadas son las siguientes:

a) Arquitecto técnico.

b) Arquitecto funcional.

c) Arquitecto Corporativo.

3.2.1.7. Un solo arquitecto para definir cambios en la arquitectura de software

La arquitectura debe ser el producto de un arquitecto solamente, o de un pequeño grupo con un único líder claramente identificado. La arquitectura macro y en una concepción abstracta debe, en lo

posible, ser realizada por un solo arquitecto. Esto brinda un componente ágil, evitando largas discusiones.

Se recomienda utilizar el Software Architecture Analysis Method **[14]** para analizar distintas arquitecturas candidatas es un método en extremo sencillo y es muy útil para comunicar ideas. Se puede usar en lugar de métodos más complicados, como ATAM **[16]**, que deberían ser tenidos en cuenta sólo para determinados casos.

El Software Engineering Institute (SEI) **[17]** ofrece una categorización de riesgos relativos a proyectos informáticos; la misma se encuentra en **[18]**. Utilizando esta categorización, pueden descubrirse riesgos no contemplados, pero también puede ser utilizada para identificar requerimientos de calidad. Por ejemplo, en la **taxonomía**, figura "disponibilidad".

Existen prácticas sencillas, algunas figuran en **[19]**. También existen varios lineamientos dentro de muchas metodologías ágiles.

Creemos que una de las mejores fuentes sobre administración de riesgos -y que todo arquitecto debería leer- se encuentra en **[20]**.

Algunas decisiones de diseño pueden tener implicancias sobre los atributos de calidad del diseño y de la aplicación, este es otro aspecto importante a considerar en el diseño de la arquitectura **[21]**. Se debe realizar una arquitectura comprensible. La arquitectura es también un medio de comunicación y el hecho de documentar la arquitectura de manera clara, sirve como nexo entre perfiles técnicos y no técnicos. Utilizar stererotypes **[22]** de manera conveniente, puede resultar de gran ayuda.

3.2.2. Componentes de la arquitectura de desarrollo

Para la metodología de desarrollo propuesta, por tratarse de desarrollo de aplicaciones Web y manejar ambientes de procesamiento distribuido, la arquitectura de software sugerida es la de "N" capas o niveles.

3.2.2.1. Programación por capas o niveles

La **programación por capas** es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.

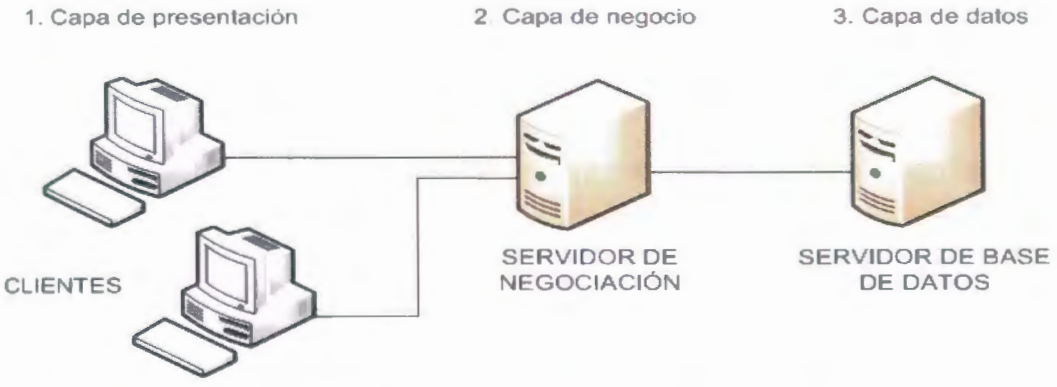


Figura 3-6 Programación por capas

En el diseño de sistemas informáticos actual se suele usar las arquitecturas multinivel o **Programación por capas**. El diseño más en boga actualmente es el diseño en tres niveles (o en tres capas).

3.2.3. Tecnologías por capas

3.2.3.1. Capa de Presentación

Las tecnologías propuestas para utilizar en la Capa de Presentación, por la compatibilidad con la visión tecnológica de crecimiento en la Superintendencia de Bancos y Seguros, son las siguientes: JSP, Servlets, Componentes UIX Oracle, Swing.

3.2.3.1.1. Soporte a la capa de presentación o vista mediante el patrón MVC en ADF

La capa de vista provee la interfase usuario a la aplicación. Oracle ADF en la capa de vista utiliza HTML , XML y Variantes, JSP, paginas UIX así como también un conjunto rico de componentes Java que hacen que la combinación de diseño y programación sea una experiencia única para el desarrollador sobretodo porque se trata de una programación declarativa en gran parte de los componentes de esta capa.

Figura 3-7 Capa de Vista

3.2.3.2. Capa de lógica de negocio

La combinación de las tecnologías Java Server Faces (JSF), ADF Business Components y Struts de Apache proporciona un sólido marco de trabajo para el desarrollo de aplicaciones Web en la capa de negocio. Para este tipo de aplicaciones se debería utilizar una arquitectura multi-capa como arquitectura de alto nivel. JSF se acopla muy bien en el patrón de diseño MVC y se puede utilizar para implementar la capa de presentación y establecer un puente de comunicación con los componentes de ADF (Business Components).

En la Superintendencia de Bancos, existe un estado de mucha inestabilidad de las aplicaciones o ya están muy caducas, los requerimientos funcionales se redefinen a diario por los usuarios finales, y la experiencia del equipo tecnológico es de alto nivel, por lo que adoptar una combinación de las tecnologías de Struts y JSF para la capa de presentación, y de ADF Business Components, representa una ventaja competitiva.

3.2.3.2.1. Soporte a la capa de negocio por los Servicios de Negocio ADF

La capa de servicios de negocio de Oracle ADF se encarga de manejar la interacción con la capa de datos, proporciona componentes para la administración de las transacciones con una fuente de datos objeto relacional, persistencia de los datos y ejecución de lógica de la aplicación o reglas de negocio.

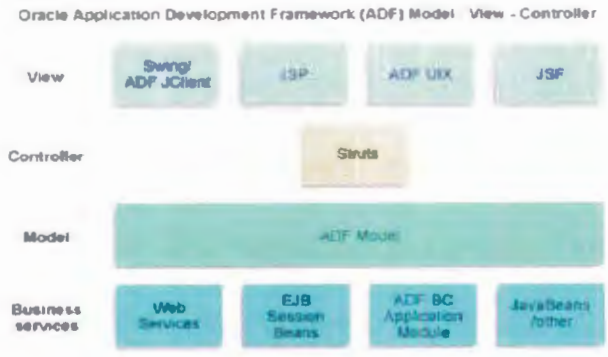


Figura 3-8 Servicios de Negocio

En la capa de servicios de negocio Oracle ADF proporciona una suite de componentes llamados "ADF Business Components" creados a partir de un modelo de datos Entidad/Relación(ER).

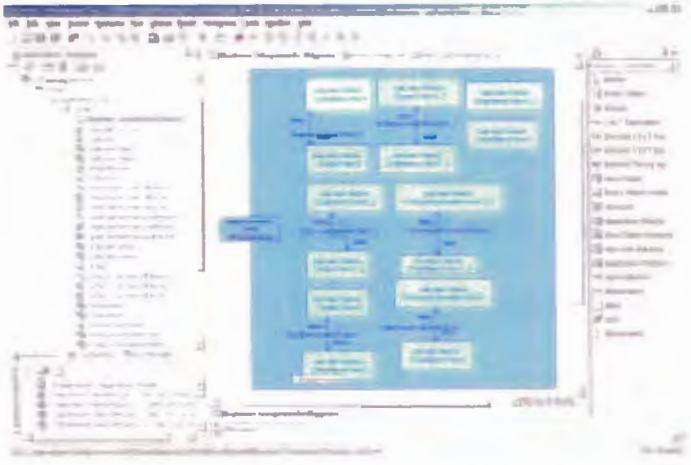


Figura 3-9 Componentes de negocio

3.2.3.3. Capa de datos

Debido a que existe un paquete completo adquirido de base de datos Oracle en la SBS, las tecnologías propuestas para la capa de datos son de Oracle, manteniendo la infraestructura existente de los servidores.

Oracle es el líder en base de datos de acuerdo a los estudios de varias empresas. Un informe Gartner 2007 indica que Oracle es la Base de Datos #1 con un 47,1% de Participación en el Mercado. Oracle obtiene el récord mundial de acuerdo con los resultados de la prueba de desempeño respecto de Windows, Linux y servidores UNIX en una gran variedad de entornos de hardware. Menor costo operativo comparado con IBM DB2 y Microsoft SQL Server.

3.2.3.3.1. Soporte a la capa modelo mediante el patrón MVC en ADF

La capa de modelo de ADF provee una interfase entre la capa

de servicios de negocio y el controlador y vista, permite conectar los objetos de la capa de servicios de negocio a con los objetos de las otras capas haciendo que la experiencia de programación sea menos tediosa.

Cuando se utiliza Oracle ADF en JDeveloper 10g puede usar la herramienta "Data Control Palette", que le permite arrastrar y soltar los controles de datos sobre páginas JSP o UIX enlazando de esta manera los controles de la capa de servicios de negocio con la interfase usuario correspondiente a la capa de vista.

Figura 3-10 Data Control Palette

3.2.3.3.2. Soporte a la capa controlador mediante el patrón MVC en ADF

La capa de controlador controla el flujo de la aplicación. Una aplicación web típica está compuesta generalmente por un gran conjunto de páginas web, en muchos casos estas se crean dinámicamente se necesita de un controlador para administrar el flujo entres estas páginas. Oracle ADF incorpora Apache Jakarta Struts como controlador, el cual es el controlador de fuente abierta más utilizado, y se ha convertido en el estándar de facto para controlar un flujo de páginas en una aplicación Web.

Figura 3-11 Flujo de páginas y Struts

3.2.3.4. Futuro de la capa de base de datos en la SBS

La próxima implementación en la capa de datos será la de Oracle Database 11*g* Enterprise Edition por la confiabilidad y escalabilidad obtenidas en la experiencia de implementar Oracle Database 10*g*.

3.2.4. Futuro de la arquitectura tecnológica de la SBS (SOA)

La arquitectura de servicios SOA es la que se empezará a utilizar al principio del año 2009 en la Superintendencia de Bancos y Seguros (SBS). Con la arquitectura SOA se propone organizar todos los recursos tecnológicos como si fueran servicios, de manera que se pueda disponer de ellos sin atender a su lógica interna.

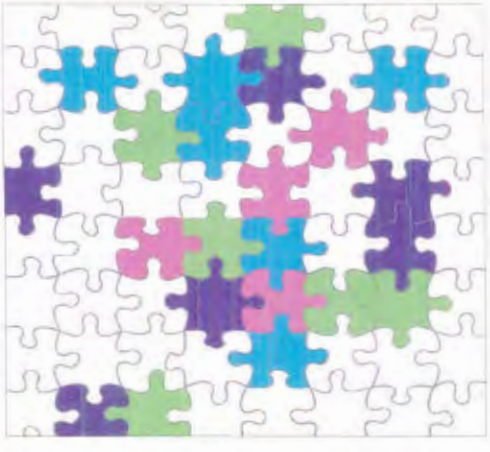


Figura 3-12 Representación de una arquitectura modular y flexible

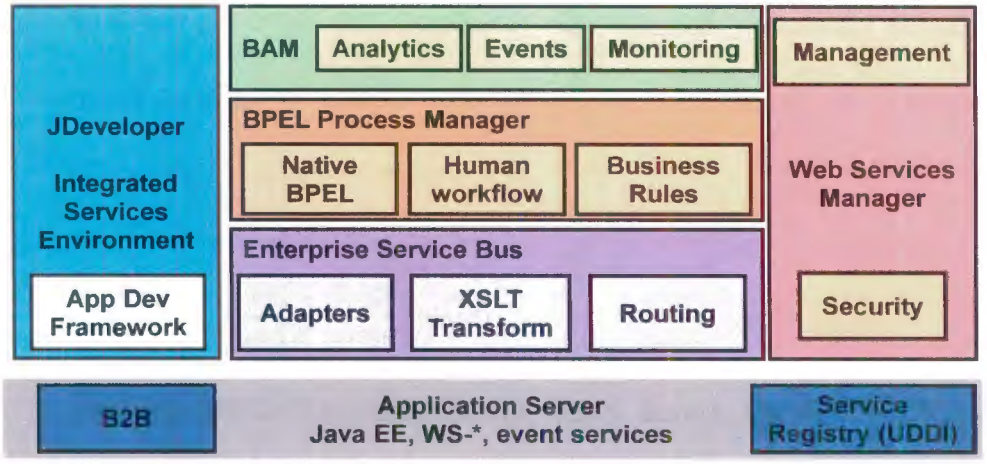


Figura 3-13 Arquitectura SOA con sus componentes

3.3. Fases de la metodología de desarrollo

3.3.1. Experiencias con el problema de estándares en empresas ecuatorianas

Durante el lapso de seis años de experiencia en el desarrollo de software en diferentes formas de negocios, he tenido que enfrentar diferentes circunstancias para construir un producto de software.

En los primeros años, me encontré con metodologías de moda para la generación de tecnologías Oracle 4GL en diferentes

empresas como por ejemplo: Banco del Pacífico, Junta de beneficencia, y éstas metodologías se basaban en desarrollos mediante un modelo en cascada.

También existieron casos de empresas como: PROCARSA, Armada Nacional, que afirmaban lo siguiente: "el desarrollo de software nos cuesta demasiado porque en la mayoría de ocasiones se hacen interminables y el desarrollo de software no es su negocio primario". Una de las razones de esta afirmación se da porque son empresas desorganizadas en el proceso de desarrollo de software, e incluso que no siguen ninguna metodología de software. A las personas encargadas del área de sistemas de estas empresas les pregunté en alguna ocasión: ¿Por qué no utilizar o adoptar una metodología de desarrollo y/o por lo menos normar ciertos procedimientos para el desarrollo de software?, y las respuestas fueron las siguientes: "existe desconfianza en los resultados de aplicación de metodologías para desarrollar, pérdida de dinero en investigación, no existe una sola metodología sino que existen muchas y no sabían cual escoger, conocían muy poco acerca de la evolución del software, no contaban con el tiempo para

implementar una metodología, y no tenían el apoyo de la jefatura del área de sistemas".

Otras empresas como: Servicios Contables del Grupo Conauto, Corporación aduanera ecuatoriana intentaban utilizar la metodología RUP, y tenían problemas para cerrar el ciclo de desarrollo de software, debido a que las iteraciones propuestas en la metodología RUP, se volvían interminables o infinitas, ya que siempre se tenían afinaciones de los requerimientos iniciales, que no establecían ningún fin en el desarrollo, al no estar claros dichos requerimientos. Había ocasiones en que este desarrollo se volvía lento por las iteraciones que se debían realizar por cada fase, y los factores externos al desarrollo del software, como por ejemplo un cambio de políticas de gobierno que se daban justo antes de terminar el proyecto, hacían que se tenga que reformular ciertas partes del software y en ocasiones tocaba realizar re-ingenierías de partes importantes del software realizado.

Por otro lado, existen empresas como Porta, Ecuatoriano Suiza, que adoptaban cierta mezcla de características de diferentes metodologías de desarrollo ágiles, dependiendo de la complejidad en el desarrollo de un software.

Adicionalmente tras leer algunos libros sobre metodologías de desarrollo durante los últimos tres años, he llegado a tener la idea: "los conceptos diversos que se manejan en las diferentes metodologías, son algo que está todavía en discusión, para que luego se pueda adoptar alguna metodología en particular". Hay montones de ellas y cada "gurú" tiene la suya. Sin embargo, casi todas ellas tienen muchas cosas comunes, así que aquí trataré de resumir la conclusión a la que he llegado. El resultado es cosa mía, recogiendo lo que he creído mejor de lo que he leído, de las experiencias en los desarrollos de software en los que he participado, para de esta manera tratar de obtener una metodología que resulte práctica.

Finalmente en la Superintendencia de Bancos y Seguros se presenta una aplicación desordenada de ciertos patrones propuesto por la metodología RUP, también no tienen estándares de codificación centralizado, y muchos problemas con la administración de la arquitectura de software y de hardware. Para este trabajo se propone el uso de estándares aplicados a una metodología de desarrollo, los problemas de la administración de la arquitectura de software y de hardware es tema de otro estudio.

3.3.2. Definición de las fases de la metodología

Luego de haber citado algunas experiencias y recogiendo la información de los conceptos comunes que se manejan en las diferentes metodologías de desarrollo de software, surgen las fases recomendadas que componen a la metodología propuesta.

Las fases de la metodología para desarrollar un proyecto de software, son básicamente las siguientes:

- Análisis de requerimientos,
- Modelado conceptual,
- Elaboración o codificación de componentes,
- Pruebas e implementación.

La fase de análisis de requerimientos, se divide en las siguientes sub-fases: El levantamiento de requisitos basado en ingeniería de requisitos y la de análisis de requerimientos.

La fase de modelado conceptual trata de establecer la arquitectura de nuestro programa. La arquitectura es un esquema de los módulos/paquetes en los que se va a dividir nuestro programa, por ejemplo qué librerías vamos a reutilizar. Para ello se deben utilizar los diagramas y documentación que sugiero para esta fase.

La fase de elaboración o codificación de componentes se refiere a la construcción de los programas, partiendo de un análisis de

requerimientos y modelo conceptual, que se encuentren validados por el usuario de uso final del software.

La fase de pruebas e implementación, se refiere a la etapa de instalación del software en un ambiente de pruebas seguido de la ejecución del plan de pruebas respectivo, para realizar la correspondiente evaluación del software entre el personal de sistemas junto al usuario final, y finalmente realizar la instalación definitiva en el ambiente de producción.

3.3.2.1. Mapa estático del negocio y condicionamiento de los objetivos

La ingeniería del software comprende que su participación en un proyecto comienza cuando existe un problema a resolver aplicando las técnicas y metodologías de esta rama del conocimiento. Sabe que este problema existe en un entorno específico, el cual, de algún modo, condiciona los resultados y las soluciones a proporcionar. Surge entonces la cuestión sobre cómo abordar la

comprensión de ese entorno. En la actualidad, no se destina tiempo del proyecto de software para comprender la organización dentro de la cual se implantará dicho software, si no que se va aprendiendo a medida que se interactúa con ella una vez que el proyecto está en curso. Se presenta como solución una protofase que articula un conjunto de técnicas y herramientas asociadas para lograr la comprensión del negocio.

3.3.2.1.1. Mapa estático del negocio

Este mapa integra la información que se obtiene a partir de las siguientes técnicas: descripción del escenario actual que implica la definición de la misión, los objetivos y las estrategias **[24]**, especificación de productos o servicios, clientes que atiende, competencia **[25]**, glosario de términos del negocio **[23]**.

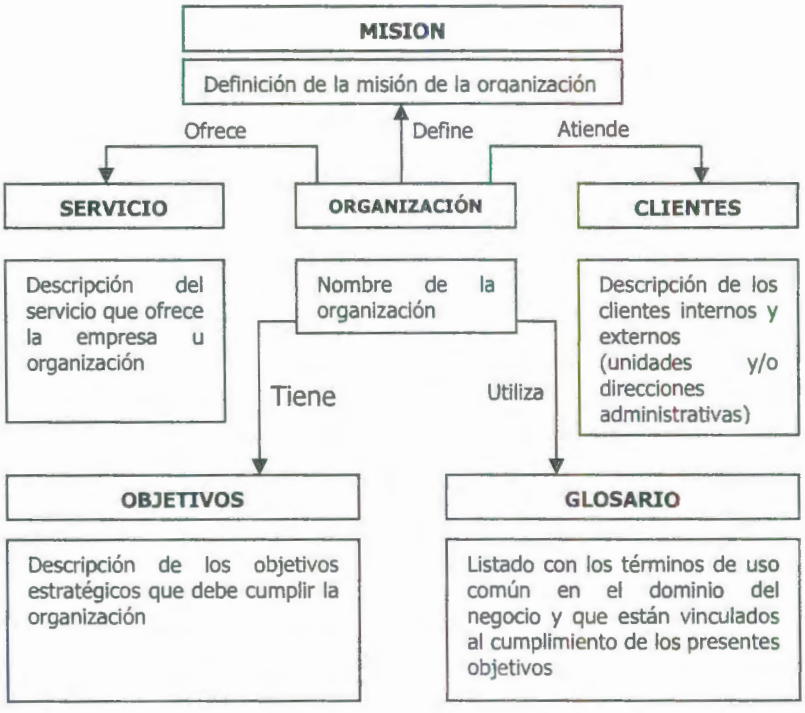


Figura 3-14 Mapa estático del negocio

3.3.2.1.2. Mapa de condicionamiento de los objetivos

Este mapa sintetiza la información que se obtiene a partir de las siguientes técnicas: análisis FODA **[26]**, análisis de los factores críticos del éxito **[27]**, análisis de riesgo **[28]** y definición del plan de contingencias **[29]**. El mapa de condicionamientos de los objetivos (Figura 3-15) presenta un carácter más dinámico que el anterior, si bien incluye el nombre de la organización y la

misión que son datos estáticos, son necesarios para dar sentido de los objetivos. Este mapa se centra en los objetivos que la organización se plantea en la actualidad, aunque estos son cambiantes. Las fortalezas, oportunidades, debilidades y amenazas son una radiografía del aquí y ahora. Lo mismo ocurre con los riesgos y los factores críticos del éxito.

Figura 3-15 Mapa de condicionamiento de los objetivos

3.3.2.1.3. Mapa táctico para alcanzar los objetivos

Este mapa (Figura 3-16) sintetiza la información de todos aquellos medios que resultan necesarios para alcanzar los objetivos, a saber: recursos humanos, fuentes de información, requerimientos **[29]**, y aquellos que se generan a partir de estos: expectativas **[30]** y restricciones **[31]**.

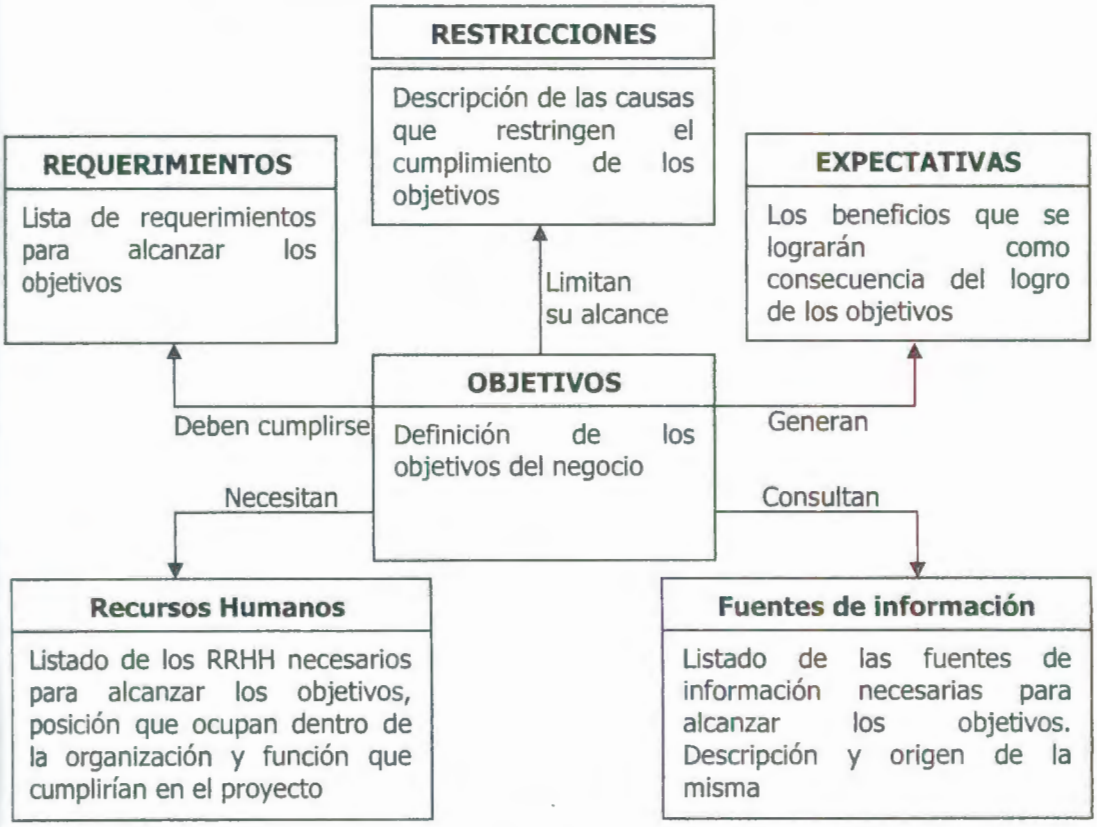


Figura 3-16 Mapa táctico para alcanzar los objetivos

3.3.2.1.4. Técnicas y Meta técnicas

En la tabla III se presenta un cuadro con aquellas técnicas y meta técnicas asociadas a las primeras, que resultan necesarias para poder alcanzar el entendimiento del negocio **[32]**. Llamamos técnicas a aquellas que generan la salida esperada, y meta técnicas a aquellas que sirven de soporte para las primeras.

Las técnicas descritas requieren de ciertas técnicas de relevamiento. En la tabla III se sugieren meta técnicas que pueden ser de utilidad para complementar los datos de salida de las técnicas propuestas necesarias para el entendimiento del negocio.

TÉCNICAS		META-TÉCNICAS					
		Entrevista Abierta	Entrevista Estructurada	JAD	Estudio de documentación	Cuestionarios	Análisis estructural de textos
Descripción del escenario actual	*Misión*	❖	❖				
	Objetivos		❖	❖			
	Estrategias		❖	❖			
	Producto / Servicio	❖			❖	❖	
	Clientes	❖				❖	
Glosario de términos		❖		❖			❖
Organigrama					❖	❖	
Análisis FODA				❖			
Análisis de los FCE				❖			
Análisis de Riesgos				❖			
Requerimientos			❖	❖			❖
Expectativas			❖	❖			
Restricciones			❖	❖			
Recursos Humanos			❖	❖			
Fuentes de Información					❖		

Tabla III. Planilla de técnicas y meta técnicas

Son aportes de este trabajo a la ingeniería de desarrollo del software:

- Señalar la falta de técnicas y herramientas para alcanzar el entendimiento del negocio dentro de las metodologías de ingeniería del software.

- Proponer una fase con identificación de técnicas y herramientas para lograr el entendimiento del negocio que puede ser incorporada como protofase (fase cero) en las metodologías de desarrollo de software de gestión (Métrica III), de desarrollo de sistemas basados en conocimiento (IDEAL) y de desarrollo de sistemas basados en explotación de información(CRISP – DM).

3.3.2.2. Análisis de requerimientos

3.3.2.2.1. Objetivos de la fase

Los objetivos de fase de análisis de requerimientos son los siguientes:

- Dar las pautas metodológicas para hacer una investigación eficaz de las necesidades.

- Capturar los requisitos de usuarios para el desarrollo del sistema.

- Desarrollar las habilidades interpersonales para obtener los requisitos de sistemas o servicios a través de la confianza mutua, entender los problemas del otro, resolver los conflictos y alcanzar el acuerdo que conduzca a un éxito compartido.

- Describir un modelo del sistema utilizando un lenguaje intermedio entre los usuarios y desarrolladores.

- Utilizar un lenguaje más formal para refinar detalles relativos a los requisitos del sistema.

- Razonar más sobre los aspectos internos del sistema.

- Estructurar los requisitos de un modo que facilite su comprensión, desarrollo, modificación, y en general, su mantenimiento.

- Identificar a los responsables de cada una de las unidades de la SBS implicadas y a los principales usuarios implicados.

- Otro objetivo de esta fase consiste en la descripción de un sistema, en términos semi-formales, que se quiere desarrollar como un proyecto de ingeniería del software orientado a objetos.

- Un objetivo más concreto de esta práctica es el de afianzar el uso de las notaciones propuestas en UML, elaborando el Modelo de Casos de Uso con los elementos UML que lo componen, mediante la realización del flujo de trabajo del proceso que se está analizando.

- También se quiere afianzar el enfoque meta-metodológico propio del Paradigma de la Orientación a Objetos, dado que se debe producir un sensible cambio de los esquemas mentales del ingeniero en el proceso de conocer y modelar un sistema de software orientado a objetos.

- En las reuniones con el cliente un objetivo es obtener información que se encuentra repartida entre varias personas, tomar decisiones estratégicas, tácticas u operativas, transmitir ideas sobre un determinado tema, analizar nuevas necesidades de información, o bien comunicar los resultados obtenidos como consecuencia de un estudio.

- Otro objetivo de esta fase es analizar y documentar las necesidades funcionales que deberán ser soportadas por el sistema a desarrollar, definiendo en forma clara, precisa, completa y verificable todas las funcionalidades y restricciones del sistema que se desea construir. Para ello

definimos en esta metodología una plantilla para documentar los requisitos funcionales, la que se revisará en secciones posteriores.

- Además de identificar los requisitos se deberán establecer las prioridades, lo cual proporciona un punto de referencia para validar el sistema final que compruebe que se ajusta a las necesidades del usuario. Las prioridades se establecen de acuerdo a las disposiciones de la autoridad de turno de la SBS.

- Mediante la técnica de entrevistas con el usuario se espera extraer los conocimientos de éste. Se deben realizar entrevistas de tipo abierta y estructurada. Este tema también será revisado en secciones posteriores.

- Detectar conflictos en los requisitos para evitar que aparezcan después, cuando es más caro resolverlos. Para

ello también se define en esta metodología una plantilla para documentar los conflictos encontrados con el usuario, y será revisada en secciones posteriores.

3.3.2.2.2. Ingeniería de requisitos

El análisis de sistemas obtiene una especificación de los sistemas, pero aquí nace la siguiente interrogante:

¿Cómo puede asegurarse que se ha especificado un sistema que recoge realmente las necesidades del usuario y satisface sus expectativas?

No hay una respuesta segura, pero la mejor solución es llevar a cabo un proceso de ingeniería de requisitos durante el análisis de un sistema.

"La ingeniería de requisitos es el procedimiento adecuado para comprender lo que quiere el usuario:

- Analizando necesidades,
- Confirmando su viabilidad,
- Negociando una solución razonable,
- Especificando la solución sin ambigüedad,
- Validando la especificación y gestionando los requisitos, para que se transformen en un sistema operacional" (Thayer & Dorfman, "Software Requirements Engineering", IEEE Press, 1997).

3.3.2.2.3. Levantamiento y negociación de los requisitos

Los requisitos son una lista de cosas que queremos que haga nuestro programa. Lo normal es que recopilemos dicha lista hablando con todas las personas que podamos: usuarios de nuestro programa, expertos en el tema de que trata el programa entre otras cosas que serán temas de discusión en las siguientes secciones.

3.3.2.2.4. Gestión de los requisitos de un proyecto de desarrollo de software

En la actualidad persisten problemas en el desarrollo de software, entre ellos, un inadecuado entendimiento de las necesidades de los usuarios, incapacidad de absorber cambios en los requisitos e insatisfacciones de los clientes por inaceptable o bajo desempeño del software. Las principales causas son la administración insuficiente de requisitos; los problemas que afectan la comunicación; las inconsistencias no detectadas entre requisitos, diseño y programación; las validaciones tardías de requisitos; el enfrentamiento reactivo de riesgos y la propagación de cambios sin control.

Las principales causas de estos problemas son la administración insuficiente de requisitos, comunicación ambigua e imprecisa, inconsistencias no detectadas entre requerimientos, diseño y programación, validaciones tardías de los requisitos,

enfrentamiento tardío de riesgos y propagación de cambios sin control **[38]**, **[36]**.

La relación no lineal entre la ingeniería de requisitos y el resto del ciclo de vida del desarrollo del software ha sido detectada desde antaño y propuestas metodológicas como el Modelo en Espiral **[33]** y el Proceso Unificado de Racional **[37]**, incorporan estrategias iterativas dentro de sus procesos de desarrollo para facilitar la ejecución de actividades propias de la ingeniería de requisitos, una vez iniciado el resto del proceso de desarrollo, al detectarse en éste la necesidad de renegociar algunos requisitos de difícil implementación o porque aparecen nuevos requisitos durante el proceso de desarrollo, entre otros. Debe tenerse en cuenta que la IR continúa durante todo el proceso de desarrollo **[40]**.

En **[34]** se ofrece una definición muy precisa de **requisito**, se dice que es la característica del sistema que es una condición para su aceptación por el cliente.

3.3.2.2.5. Desarrollo de requisitos

Los principios para realizar la gestión de los requisitos del software son:

- El acuerdo de los requisitos es el puente entre el desarrollo de requisitos y la gestión de requisitos.

- La gestión de requisitos incluye todas las actividades para mantener la integridad, exactitud y difusión de los acuerdos de los requisitos durante la vida del proyecto.

Los cambios deben controlarse y documentarse. Hay que convivir con el cambio. Por lo tanto, es esencial planear posibles cambios a los requerimientos cuando el sistema sea desarrollado y utilizado. Por tanto la gestión de los requisitos del software, de su evolución es un proceso externo que ocurre a lo largo del ciclo de vida del proyecto.

3.3.2.2.6. Procedimiento de gestión de cambios en los requisitos

Desde el inicio hay que establecer la conformación de la línea base de requisitos, como un canal simple para el control de cambios, que se podrá usar para "capturar" nuevos cambios.

Para ello hay que tener claro desde que se obtiene el primer requisito de un software a desarrollar, lo siguiente:

1. **Línea base de requisitos:** Se considera a los requisitos de línea de base a los que se definen en una primera iteracción utilizando las tareas el procedimiento para la administración de requisitos de la sección 3.3.2.2.8.

2. **Canal y control de cambios:** En vista que las peticiones de cambios provienen de muchas fuentes, las mismas deben ser enrutadas en un solo proceso. Esto se hace con la

finalidad de evitar problemas y conseguir estabilidad en los requerimientos. Obsérvense las figuras 3-17 y 3-18.

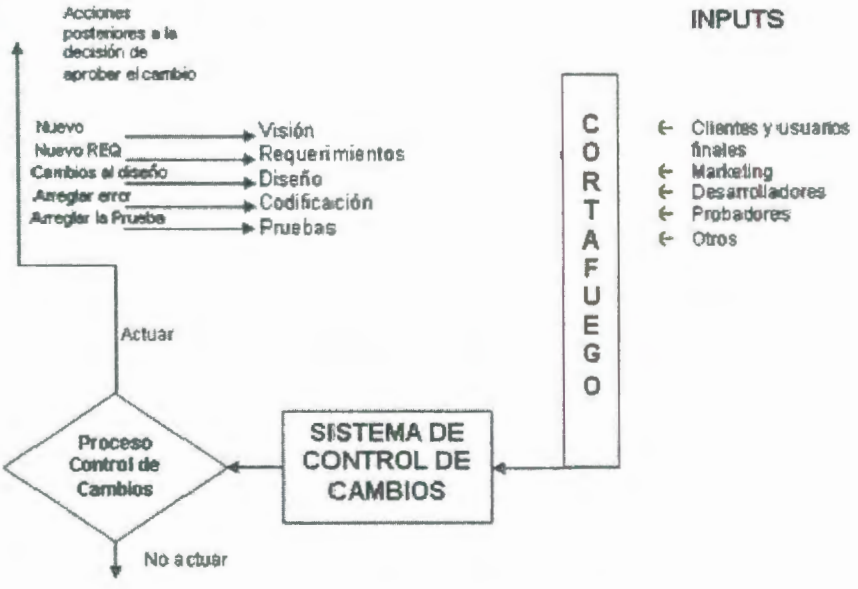


Figura 3-17 Canal de cambios.

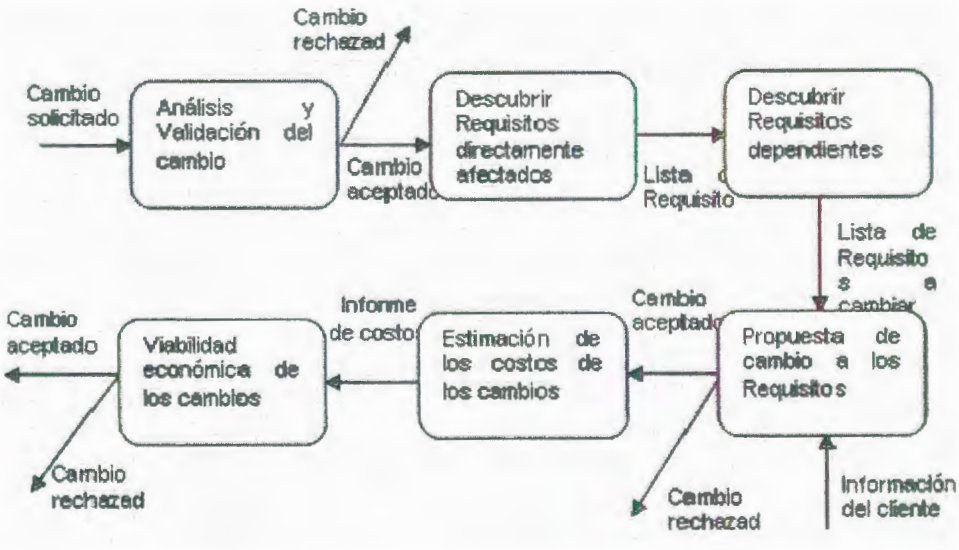


Figura 3-18 Proceso de control de cambios.

Las solicitudes de cambios, desde el momento en que son comunicadas hasta su implementación, transitan por diferentes estados en el que intervienen los implicados asumiendo diferentes roles. Obsérvense la figura 3-19 y la tabla IV.

El canal de cambios y el proceso de control de cambios serán aprobados por el departamento de políticas y de seguridades de la Dirección Nacional de Recursos Tecnológicos de la SBS. Se menciona en esta sección este proceso de control de cambios, recomendando al departamento de políticas y de seguridades de la Dirección Nacional de Recursos Tecnológicos para que se elabore la política respectiva.

3. **Ejecución de los cambios:** Aprobado el cambio se procede a su implementación, de acuerdo a la fase del proyecto a que corresponda. En caso de que los cambios aprobados:

- Impliquen el desarrollo de un nuevo sistema, entonces será necesario comenzar un nuevo proceso de IR.

- Impliquen la implementación de nuevos requisitos, entonces será necesario comenzar por la actividad de elicitación o educción de requisitos.

- afecten otras fases del proceso de desarrollo del proyecto, entonces se implementan en esas.

4. **Trazabilidad:** Los requisitos deben ser "rastreables": por su origen (¿quién lo propuso?); necesidad (¿por qué existe?); por su relación con otros requisitos; por su relación con elementos del diseño y/o la implementación. Esta información es útil para saber qué afecta un cambio en un requisito. Se recomienda que esta información quede archivada en el departamento de políticas, organizada por cada nuevo desarrollo de una aplicación Web.

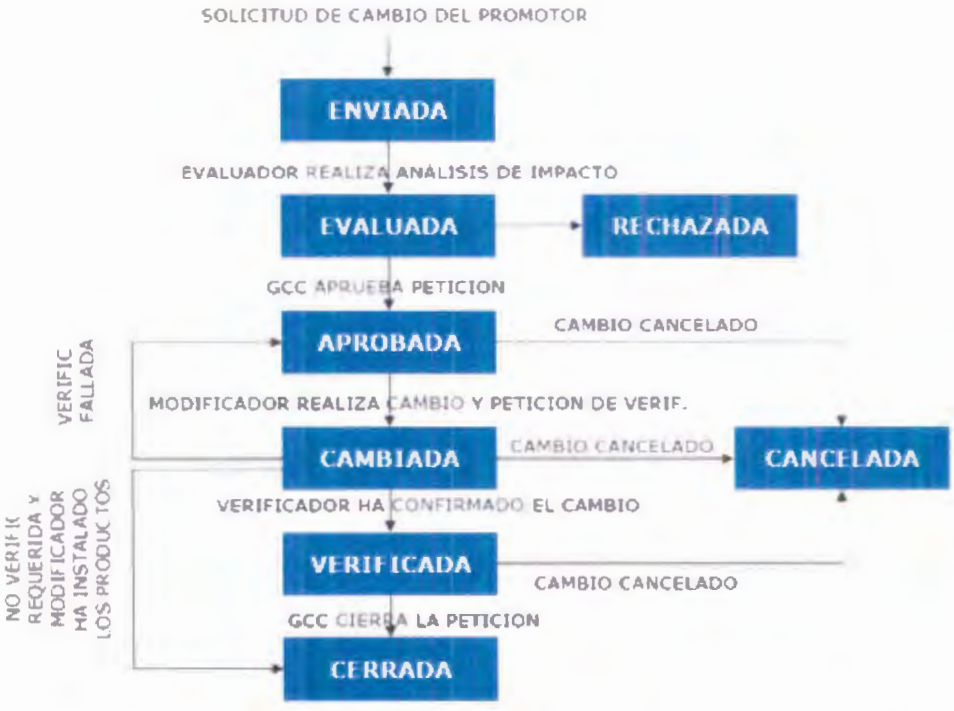


Figura 3-19 Estados de una petición de cambio

Rol	Descripción
Grupo de control de cambios (GCC)	Grupo de personas que deciden aprobar o rechazar las peticiones de cambios para un proyecto específico.
Promotor del cambio	Persona autorizada que solicita la petición de cambio de requisitos.
Evaluador	Persona que analiza el impacto de la petición de cambio (puedes ser técnico, martketing, cliente o combinación).
Modificador	Persona que realiza el cambio como consecuencia de una petición de cambio aprobada.
Verificador	Persona que determina si el cambio se ha realizado correctamente.
Validador	Persona del cliente que valida la implementación del cambio realizado.

Tabla IV. Roles en el procedimiento de control de cambios

En **[35]** la trazabilidad está definida como una técnica usada para "proveer una relación entre requisitos, el diseño y la implementación final del sistema". En **[39]** se establece que "la trazabilidad da asistencia esencial en el entendimiento de las relaciones que existen entre y a través los requisitos, el diseño y la implementación".

Estas relaciones pueden ser rastreables con la documentación generada en el Procedimiento para la administración de requisitos de la sección 3.3.2.2.8.

5. **Control de versiones:** El control de versiones de documentos, diagramas, programas, y toda la documentación generada en el desarrollo de un proyecto de software, se la realizará con los software: WinCVS y OracleCVS respectivamente.

3.3.2.2.7. Buenas prácticas de la actividad de gestión de requisitos

Las buenas prácticas en la actividad de gestión de requisitos que se implementarán en la SBS son las siguientes:

- Definir un proceso de control de cambios, para el caso de la SBS se recomienda establecer como política el control de versiones.

- Establecer un grupo (o comité) de control de cambios en el departamento de políticas y de seguridades de la Dirección Nacional de Recursos Tecnológicos de la SBS.

- Realizar análisis del impacto sobre los cambios por parte del comité.

- Creación de las líneas base y controlar las versiones de los requisitos, por parte del departamento de políticas y de seguridades.

El proyecto puede responder frente a petición de nuevos requisitos o petición de cambios en los requisitos de diferentes maneras:

- Diferir los requisitos de baja prioridad, esto será responsabilidad de la subdirección de desarrollo y aplicaciones tecnológicas (SDAT).

- Ordenar realizar más horas de trabajo (preferiblemente durante un periodo corto de tiempo).

- Retrasar el calendario para acomodarse a la nueva funcionalidad.

- Permitir reducir el nivel de calidad bajo la presión de mantener la fecha original (frecuentemente esta es la reacción por defecto).

En base a las recomendaciones de la gestión de requisitos se propone las siguientes actividades y tareas que se enmarcan en un **procedimiento para la administración de requisitos**, tomando en consideración que se trata de un procedimiento que dura mientras se cumple el ciclo de vida del software.

3.3.2.2.8. Procedimiento para la administración de requisitos

El objetivo de este procedimiento es la de definir las tareas a realizar, los productos a obtener y las técnicas a emplear durante la sub-etapa de *levantamiento de requisitos* de la fase de *análisis de requerimientos* del desarrollo de software. El único producto entregable definido en este procedimiento es el *Documento de Requisitos del Software* (DRS). Las tareas del

procedimiento recomendadas para obtener los productos descritos son las siguientes:

Tarea 1: Obtener información sobre el dominio del problema y el sistema actual.

Tarea 2: Preparar y realizar las reuniones de investigación/negociación de requisitos.

Tarea 3: Identificar y revisar los objetivos del sistema.

Tarea 4: Identificar y revisar los requisitos de almacenamiento de información.

Tarea 5: Identificar y revisar los requisitos funcionales.

Tarea 6: Identificar y revisar los requisitos no funcionales.

Tarea 7: Priorizar objetivos y requisitos.

El orden recomendado de realización para estas tareas es: 1...7, aunque las tareas 4, 5, y 6 pueden realizarse simultáneamente o en cualquier orden que se considere oportuno (ver figura 3-20). En las siguientes secciones se describen cada una de las tareas mencionadas.

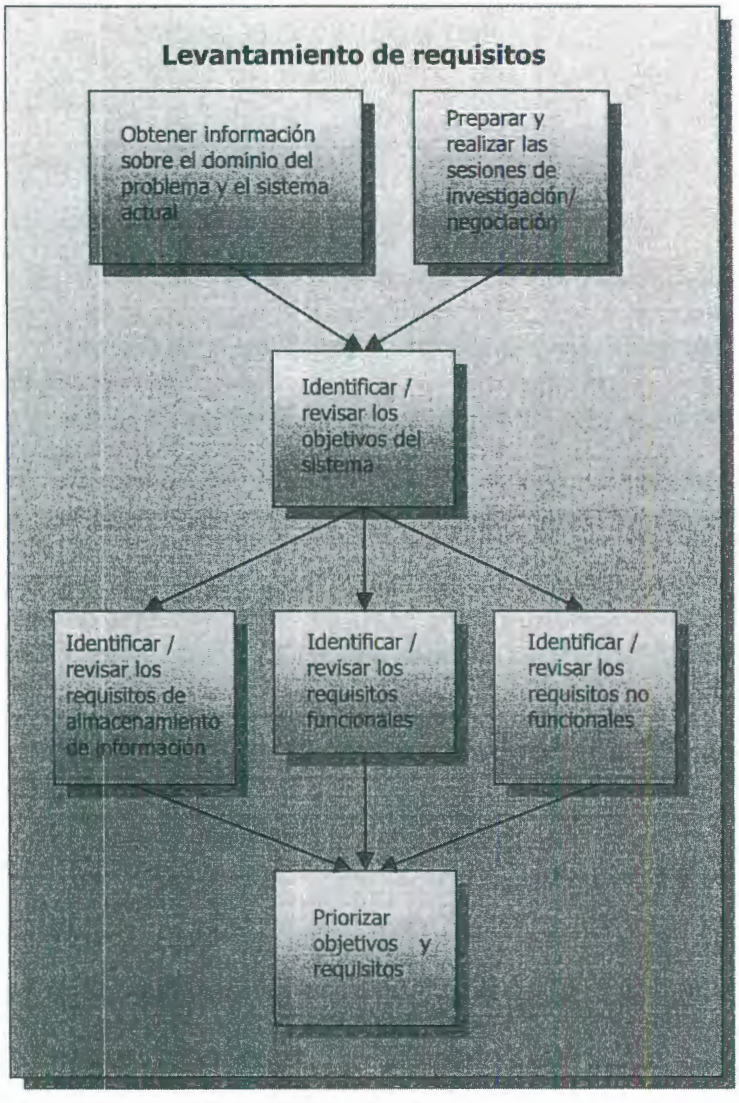


Figura 3-20 Tareas de levantamiento/investigación de requisitos

3.3.2.2.8.1. Tarea 1: Obtener información sobre el dominio del problema y el sistema actual

Los objetivos de esta tarea son:

- Conocer el dominio del problema.
- Conocer la situación actual.

Esta tarea es opcional, ya que puede que no sea necesario realizarla si el equipo de desarrollo tiene experiencia en el dominio del problema y el sistema actual es conocido.

Los productos internos de esta tarea son los siguientes:

- Información recopilada: libros, artículos, folletos comerciales, desarrollos previos sobre el mismo dominio, etc.

- Modelos del sistema actual.

Los productos entregables son: Introducción, participantes en el proyecto, principalmente clientes y desarrolladores, y descripción del sistema actual como parte del DRS.

Entre las técnicas recomendadas de esta tarea se mencionan:

- Obtener información de fuentes externas al negocio del cliente: folletos, informes sobre el sector, publicaciones, consultas con expertos, etc.

- En el caso de que se trate de un dominio muy específico puede ser necesario recurrir a fuentes internas al propio negocio del cliente, en cuyo caso pueden utilizarse las técnicas auxiliares de levantamiento o investigación de requisitos como el estudio de documentación, observación *in situ*, cuestionarios, inmersión o *aprendizaje*, etc.

- Modelado del sistema actual [Laguna et *al.* 1999, García et *al.* 2000].

3.3.2.2.8.2. Tarea 2: Preparar y realizar las sesiones de investigación/negociación

Los objetivos de esta tarea son:

- Identificar a los usuarios participantes.
- Conocer las necesidades de clientes y usuarios.
- Resolver posibles conflictos.

Esta tarea es especialmente crítica y ha de realizarse con especial cuidado, ya que generalmente el equipo de desarrollo no conoce los detalles específicos de la organización o unidad administrativa de la SBS para la que se va a desarrollar el sistema y, por otra parte, los clientes y posibles usuarios no saben qué necesita saber el equipo de desarrollo para llevar a cabo su labor.

Los productos internos de esta tarea son: notas tomadas durante las reuniones, transcripciones o actas de reuniones,

formularios, grabaciones en cinta o vídeo de las reuniones o cualquier otra documentación que se considere oportuna.

Los productos entregables de la tarea son los siguientes:

- Participantes en el proyecto, en concreto los usuarios participantes, como parte del DRS.

- Objetivos, requisitos o conflictos, que se hayan identificado claramente durante las sesiones de investigación/negociación, como parte del DRS.

Las técnicas recomendadas a utilizar en esta tarea son:

- Técnicas de investigación de requisitos, incluyendo las plantillas de objetivos, requisitos y conflictos descritas en la sección de plantillas más adelante, que pueden usarse

directamente durante las sesiones de investigación/negociación.

- Técnicas de negociación como *WinWin* [Boehm *et al.* 1994].

3.3.2.2.8.3. Tarea 3: Identificar/revisar los objetivos del sistema

Los objetivos de esta tarea son:

- Identificar los objetivos que se esperan alcanzar mediante el sistema/software a desarrollar.

- Revisar, en el caso de que haya conflictos, los objetivos previamente identificados.

A partir de la información obtenida en la tarea anterior, en esta tarea se deben identificar qué objetivos se esperan

alcanzar una vez que el sistema/software a desarrollar se encuentre en explotación o revisarlos en función de los conflictos identificados. Puede que los objetivos hayan sido proporcionados antes de comenzar el desarrollo.

No hay productos internos en esta tarea.

El producto entregable de esta tarea es definir los objetivos del sistema como parte del DRS, y la estructura del Documento de Requisitos del Sistema se verá más adelante.

Las técnicas recomendadas para realizar esta tarea son:

- Análisis de factores críticos de éxito (FCE) [MAP 1995] o alguna técnica similar de identificación de objetivos.

3.3.2.2.8.4. Tarea 4: Identificar/revisar los requisitos de almacenamiento de información

Los objetivos de esta tarea son los siguientes:

- Identificar los requisitos de almacenamiento de información que deberá cumplir el sistema/software a desarrollar.

- Revisar, en el caso de que haya conflictos, los requisitos de almacenamiento de información previamente identificados.

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados en la tarea 3 y el resto de los requisitos, en esta tarea se debe identificar, o revisar si existen conflictos, qué información relevante para el cliente deberá gestionar y almacenar el sistema/software a desarrollar.

No hay productos internos en esta tarea.

El producto entregable de esta tarea es la definición de los requisitos de almacenamiento de información como parte del DRS.

La técnicas recomendada es la de utilizar la plantilla para requisitos de almacenamiento de información.

3.3.2.2.8.5. Tarea 5: Identificar/revisar los requisitos funcionales

Los objetivos de esta tarea son los siguientes:

- Identificar los actores del sistema/software a desarrollar.
- Identificar los requisitos funcionales (casos de uso) que deberá cumplir el sistema/software a desarrollar.

Revisar, en el caso de que haya conflictos, los requisitos funcionales previamente identificados.

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados en la tarea 3 y el resto de los requisitos, en esta tarea se debe identificar, o revisar si existen conflictos, qué debe hacer el sistema a desarrollar con la información identificada en la tarea anterior. Inicialmente se identificarán los actores que interactuarán con el sistema, es decir aquellas personas u otros sistemas que serán los orígenes o destinos de la información que consumirá o producirá el sistema a desarrollar y que forman su entorno.

A continuación se identificarán los casos de uso asociados a los actores, los pasos de cada caso de uso y posteriormente se detallarán los casos de uso con las posibles excepciones hasta definir todas las situaciones posibles.

No hay productos internos en esta tarea.

El producto entregable de esta tarea es la definición de los requisitos funcionales como parte del DRS.

Las técnicas recomendadas a utilizar en esta tarea son las siguientes:

- Casos de uso.
- Plantilla para actores.
- Plantilla para los requisitos funcionales.

3.3.2.2.8.6. Tarea 6: Identificar/revisar los requisitos no funcionales

El objetivo de esta tarea es la de identificar los requisitos no funcionales del sistema/software a desarrollar.

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados en la tarea 3 y

el resto de los requisitos, en esta tarea se deben identificar, o revisar si existen conflictos, los requisitos no funcionales, normalmente de carácter técnico o legal.

Algunos tipos de requisitos que se suelen incluir en esta sección son los siguientes:

a) Requisitos de comunicaciones del sistema.

b) Requisitos de interfaz de usuario.

c) Requisitos de fiabilidad.

d) Requisitos de entorno de desarrollo.

e) Requisitos de portabilidad.

No hay productos internos en esta tarea.

El producto entregable de esta tarea es la definición de los requisitos no funcionales del sistema como parte del DRS.

La técnica recomendada para esta tarea es la de uso de la plantilla para requisitos no funcionales.

El único producto entregable que se contempla en este procedimiento es el *Documento de Requisitos del Software* (DRS).

3.3.2.2.8.7. Tarea 7: Priorizar los objetivos y requisitos

En esta tarea se definen las prioridades de los objetivos y de los requisitos del nuevo software, y como se recomendación tanto los objetivos como los requisitos del sistema deben estar alineados a la estrategia institucional de la SBS. La Subdirección de Desarrollo Institucional de la SBS (SDI), tiene definidas las estrategias institucionales de la SBS con fechas de vigencia, y por cada estrategia están definidos ciertos grupos de objetivos estratégicos, por lo que resulta conveniente que el Subdirector de Desarrollo de Aplicaciones Tecnológicas realice un estudio exhaustivo de las prioridades

de los objetivos y requisitos alineado al soporte de la estrategia institucional de la SBS.

3.3.2.2.8.8. Estructura del Documento de Requisitos del Software

La estructura del DRS puede verse en la figura 3-21. A continuación se describe con detalle cada sección del DRS.

1. **Portada:** La portada del DRS debe tener el formato que puede verse en la figura 3-22. Los elementos que deben aparecer son los siguientes:

   a) **Nombre del proyecto:** el nombre del proyecto al que pertenece el DRS.

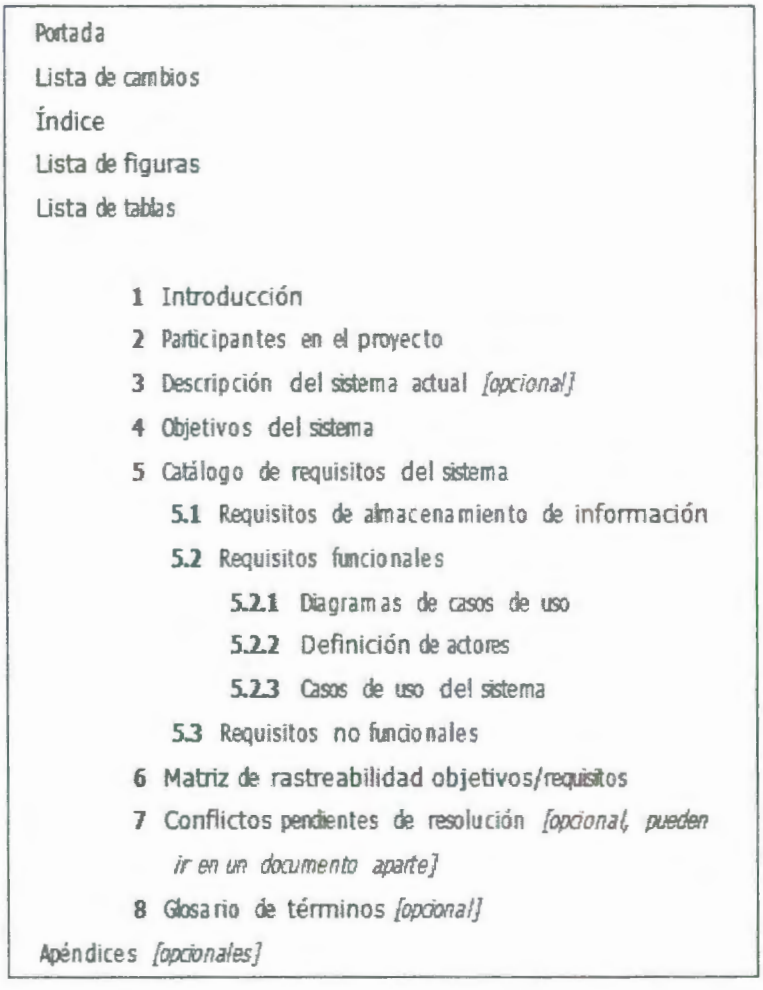


Figura 3-21 Estructura del Documento de Requisitos del Sistema

b) Versión: la versión del DRS que se entrega al cliente. La versión se compone de dos números *X* e *Y*. El primero indica la versión, y se debe incrementar cada vez que se hace una nueva entrega formal al cliente. Cuando se incremente el primer número, el segundo debe volver a comenzar en cero. El segundo número

indica cambios dentro de la misma versión aún no entregada.

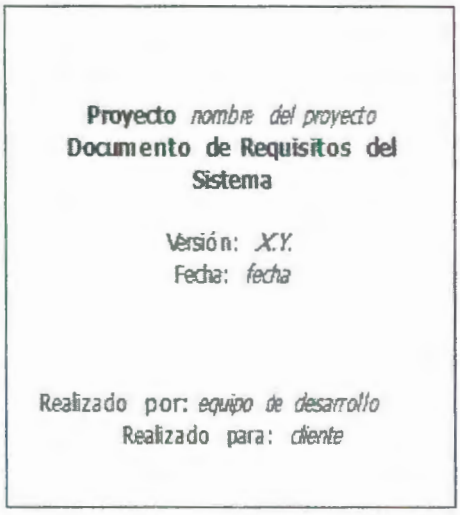


Figura 3-22 Portada del Documento de Requisitos del Sistema

c) **Fecha:** fecha de la publicación de la versión.

d) **Equipo de desarrollo:** nombre de la empresa o equipo de desarrollo.

e) **Cliente:** nombre del cliente, normalmente otra unidad de la SBS.

2. **Lista de cambios:** el documento debe incluir una lista de cambios en la que se especifiquen, para cada versión

del documento, los cambios producidos en el mismo con un formato similar al que puede verse en la figura 3-23.

3. **Índice:** el índice del DRS debe indicar la página en la que comienza cada sección, sub-sección o apartado del documento. En la medida de lo posible, se sangrarán las entradas del índice para ayudar a comprender la estructura del documento.

Núm.	Fecha	Descripción	Autores
0	$fecha_0$	Versión x.y	$autor_0$
1	$fecha_1$	$descripción\ cambio_1$	$autor_1$
⋮	⋮	⋮	⋮
n	$fecha_n$	$descripción\ cambio_n$	$autor_n$

Figura 3-23 Lista de cambios del Documento de Requisitos del Sistema

4. **Listas de figuras y tablas:** el DRS deberá incluir listas de las figuras y tablas que aparezcan en el mismo. Dichas listas serán dos índices que indicarán el número, la descripción y la página en que aparece cada figura o tabla del DRS.

5. **Introducción:** esta sección debe contener una descripción breve de las principales características del sistema/software que se va a desarrollar.

6. **Participantes en el proyecto:** esta sección debe contener una lista con todos los participantes en el proyecto, tanto desarrolladores como clientes y usuarios.

7. **Descripción del sistema actual:** esta sección debe contener una descripción del sistema actual en el caso de que se haya acometido su estudio. Para describir el sistema actual puede utilizarse cualquier técnica que se considere oportuno, por ejemplo las descritas en [Laguna *et al.* 1999] (*Diagrama Documentos-Tarea,* DDT) o en **[36]** (*Diagramas de Actividad*, también descritos en [Booch *et al.* 1999]).

8. **Objetivos del sistema:** esta sección debe contener una lista con los objetivos que se esperan alcanzar cuando el sistema/software a desarrollar esté en explotación, especificada mediante la plantilla para objetivos descrita en la sección de plantillas más adelante.

9. **Catálogo de requisitos del sistema:** esta sección se divide en las siguientes sub-secciones en las que se describen los requisitos del sistema.

10. **Matriz de rastreabilidad objetivos/requisitos:** esta sección debe contener una matriz *objetivo-requisito,* de forma que para cada objetivo se pueda conocer con qué requisitos está asociado.

	OBJ–01	OBJ–02	...	OBJ-*n*
RI–01	•	•		
RI–02		•		
...				
RF–01	•			
RF–02	•	•		

Figura 3-24 Matriz de rastreabilidad del Documento de Requisitos del Sistema

- **Conflictos pendientes de resolución:** esta sección, que se incluirá en el caso de que no se opte por registrar los conflictos en un documento aparte, deberá contener los conflictos identificados durante el proceso y que aún están pendientes de resolución, descritos mediante la plantilla para conflictos.

- **Glosario de términos:** esta sección, que se incluirá si se considera oportuno, deberá contener una lista ordenada alfabéticamente de los términos específicos del dominio del problema, acrónimos y abreviaturas que aparezcan en el documento.

- **Apéndices:** los apéndices se usarán para proporcionar información adicional a la documentación obligatoria del documento.

3.3.2.2.8.9. Técnicas recomendadas para la investigación de requisitos

A continuación, se describen algunas de las técnicas que se proponen en este procedimiento para obtener los productos de las tareas que se han descrito.

Las técnicas más habituales en la investigación o levantamiento de requisitos son las entrevistas, el *Joint Application Development* (JAD) o Desarrollo Conjunto de Aplicaciones, el *brainstorming* o tormenta de ideas y la utilización de escenarios [Weidenhaput *et al.* 1998, Rolland *et al.* 1998], más conocidos como *casos de uso* [Jacobson *et al.* 1993, Booch *et al.* 1999]..

Las técnicas recomendadas a ser utilizadas son:

1. Entrevistas.
2. Joint Application Development (Sesiones JAD).
3. Brainstorming.

3.3.2.2.8.10. Realización de prototipos de software

Se recomienda realizar prototipos de software cuando son desarrollos de tiempo muy corto, debido a las emergencias que se pudieran dar en la SBS.

Es necesario que pueda construirse más rápidamente que el sistema real. Su utilidad principal será la de superar problemas de articulación de requisitos y barreras de comunicación entre usuarios y desarrolladores.

Para construir un prototipo de software se deberá utilizar los estándares visuales propuestos en el "ANEXO-6-Estándares visuales de páginas Web".

3.3.2.2.8.11. Plantillas y patrones lingüísticos para el levantamiento de requisitos

Las plantillas y patrones lingüísticos que se presentan en los siguientes apartados están pensados para utilizarse tanto durante las reuniones de investigación con clientes y usuarios de la SBS, como para registrar y gestionar los requisitos (ver figura 3-25).

Figura 3-25 La plantilla como elemento de investigación y negociación

En la notación usada para describir los patrones-L, las palabras o frases entre < y > deben ser convenientemente reemplazadas, mientras que las palabras o frases que se encuentren entre { y } y separadas por comas representan opiniones de las que se debe escoger una.

3.3.2.2.8.12. Plantilla para los objetivos del sistema

Los objetivos del sistema pueden considerarse como *requisitos de alto nivel* **[40]**, de forma que los requisitos propiamente dichos serían la forma de alcanzar los objetivos. La plantilla propuesta para los objetivos puede verse en la figura 3-26. El significado de los campos que la componen, cuya mayoría está presente también en las plantillas para los requisitos, es el siguiente:

Identificador y nombre descriptivo: siguiendo la propuesta, entre otros, de [Sawyer *et al.* 1997], cada objetivo debe identificarse por un código único y un nombre descriptivo. Con objeto de conseguir una rápida identificación, los identificadores de los objetivos comienzan con *OBJ*.

Versión: para poder gestionar distintas versiones, este campo contiene el número y la fecha de la versión actual del objetivo.

Autores, Fuentes: estos campos contienen el nombre y la organización de los autores (normalmente desarrolladores) y de las fuentes (clientes o usuarios), de la versión actual del objetivo.

OBJ–<*id*>	<*nombre descriptivo*>
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)
Autores	• <*autor de la versión actual*> (<*organización del autor*>) ...
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) ...
Descripción	El sistema deberá <*objetivo a cumplir por el sistema*>
Subobjetivos	• OBJ–x <*nombre del subobjetivo*> • ...
Importancia	<*importancia del objetivo*>
Urgencia	<*urgencia del objetivo*>
Estado	<*estado del objetivo*>
Estabilidad	<*estabilidad del objetivo*>
Comentarios	<*comentarios adicionales sobre el objetivo*>

Figura 3-26 Plantilla y patrones-L para objetivos

Descripción: Este campo contiene un patrón-L que se debe completar con la descripción del objetivo.

Sub-objetivos: en este campo pueden indicarse los sub-objetivos que dependen del objetivo que se está describiendo.

Importancia: este campo indica la importancia del cumplimiento del objetivo para los clientes y usuarios. Se puede asignar un valor numérico o alguna expresión enumerada como *vital, importante* o *quedaría bien*, tal como se propone en [IBM OOTC 1997]. En el caso de que no se haya establecido aún la importancia, se puede indicar que está *por determinar* (*PD*), equivalente al TBD (*To Be Determined*), empleado en las especificaciones escritas en inglés.

Urgencia: este campo indica la urgencia del cumplimiento del objetivo para los clientes y usuarios en el supuesto caso de un desarrollo incremental. Como en el caso anterior, se puede asignar un valor numérico o una expresión enumerada

como *inmediatamente, hay presión* o *puede esperar* [IBM OOTC 1997], o *PD* en el caso de que aún no se haya determinado.

Estado: este campo indica el estado del objetivo desde el punto de vista de su desarrollo. El objetivo puede estar *en construcción* si se está elaborando, *pendiente de negociación* si tiene algún conflicto pendiente y está a la espera de validación o, por último, puede estar *validado* si ha sido validado por clientes y usuarios.

Estabilidad: este campo indica la estabilidad del objetivo, es decir una estimación de la probabilidad de que pueda sufrir cambios en el futuro. Esta estabilidad puede indicarse mediante un valor numérico o mediante una expresión enumerada como *alta, media* o *baja* o *PD* en el caso de que aún no se haya determinado.

Comentarios: cualquier otra información sobre el objetivo que no encaje en los campos anteriores puede recogerse en este apartado.

3.3.2.2.8.13. Plantilla para requisitos de almacenamiento de información

Lo más importante en los sistemas de información es precisamente la información que gestionan. La plantilla para requisitos de almacenamiento de información, que puede verse en la figura 3-27, ayuda a los clientes y usuarios a responder a la pregunta "*¿qué información, relevante para los objetivos de su negocio, debe ser almacenada por el sistema?*".

El significado de los campos de la plantilla es el siguiente:

Identificador y nombre descriptivo: siguiendo las recomendaciones entre otros, de [IEEE 1993] y [Sawyer *et al.*

1997]. Con objeto de conseguir una rápida identificación, los identificadores de los requisitos de almacenamiento de información comienzan con *RI.*

Versión, Autores, Fuentes: estos campos tienen el mismo significado que en la plantilla para objetivos aunque referidos al requisito.

Objetivos asociados: este campo debe contener una lista con los objetivos a los que está asociado el requisito. Esto permite conocer qué requisitos harán que el sistema a desarrollar alcance los objetivos propuestos.

Descripción: para los requisitos de almacenamiento de información este campo usa un patrón-L que se debe completar con el concepto relevante sobre el que se debe almacenar información.

RI–<*id*>	<*nombre descriptivo*>
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)
Autores	• <*autor de la versión actual*> (<*organización del autor*>) ...
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) ...
Objetivos asociados	• OBJ–*x* <*nombre del objetivo*> ...
Requisitos asociados	• Rx–y <*nombre del requisito*> ...
Descripción	El sistema deberá almacenar la información correspondiente a <*concepto relevante*>. En concreto:
Datos específicos	• <*datos específicos sobre el concepto relevante*> • ...
Intervalo temporal	{ pasado y presente, sólo presente }
Importancia	<*importancia del requisito*>
Urgencia	<*urgencia del requisito*>
Estado	<*estado del requisito*>
Estabilidad	<*estabilidad del requisito*>
Comentarios	<*comentarios adicionales sobre el requisito*>

Figura 3-27 Plantilla y patrones-L para requisitos de almacenamiento de información

Datos específicos: este campo contiene una lista de los datos específicos asociados al concepto relevante, de los que pueden indicarse todos aquellos aspectos que se considere oportunos (descripción, restricciones, ejemplos, etc.).

Intervalo temporal: este campo indica durante cuánto tiempo es relevante la información para el sistema. Puede tomar los valores *pasado* y *presente*, si la información es

siempre relevante, o sólo *presente* si la información tiene un periodo de validez concreto.

Requisitos asociados: en este campo se indican otros requisitos que estén asociados por algún motivo con el requisito que se está describiendo, permitiendo así tener una rastreabilidad *horizontal*, similar a las relaciones entre *assets* (activos) del mismo nivel descritas en **[36]**.

Importancia, Urgencia, Estado, Estabilidad, Comentarios: estos campos tienen el mismo significado que en la plantilla para objetivos aunque referidos al requisito.

3.3.2.2.8.14. Plantilla para actores

Aunque, estrictamente hablando, los actores de los casos de uso no son requisitos, por homogeneidad con el estilo de definición del resto de los elementos que componen el

catálogo de requisitos se ha descrito la plantilla para definirlos que puede verse en la figura 3-28.

El único campo específico de esta plantilla es la descripción, en la que se usa en patrón-L que debe completarse con la descripción del rol o papel que representa el actor respecto al sistema. El significado del resto de los campos es el mismo que para las plantillas anteriores.

ACT-<*id*>	<*nombre descriptivo*>
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)
Autores	• <*autor de la versión actual*> (<*organización del autor*>) ...
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) ...
Descripción	Este actor representa a <*rol que representa el actor*>
Comentarios	<*comentarios adicionales sobre el actor*>

Figura 3-28 Plantilla y patrones-L para actores

3.3.2.2.8.15. Plantilla para requisitos funcionales

Los sistemas de información no sólo almacenan información, también deben proporcionar servicios usando la información que almacenan. La plantilla de requisitos funcionales, que

puede verse en la figura 3-29, describe casos de uso y ayuda a los clientes y usuarios a responder a la pregunta "*¿qué debe hacer el sistema con la información almacenada para alcanzar los objetivos de su negocio?*".

El significado de los campos específicos de esta plantilla es el siguiente (los campos comunes con la plantilla para requisitos de almacenamiento de información tienen el mismo significado):

Identificador y nombre descriptivo: igual que en la plantilla anterior, excepto que los identificadores de los requisitos funcionales empiezan con RF y que el nombre descriptivo suele coincidir con el objetivo que los actores esperan alcanzar al realizar el caso de uso.

Descripción: para los requisitos funcionales, este campo contiene un patrón-L que debe completarse de forma distinta en función de que el caso de uso sea abstracto o concreto.

Precondición: en este campo se expresan en lenguaje natural las condiciones necesarias para que se pueda realizar el caso de uso.

Secuencia normal: este campo contiene la secuencia normal de interacciones del caso de uso. En cada paso, un actor o el sistema realiza una o más acciones, o se realiza (se incluye) otro caso de uso. Se asume que, después de realizar el último paso, el caso de uso termina.

RF–<*id*>	<*nombre descriptivo*>	
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)	
Autores	• <*autor de la versión actual*> (<*organización del autor*>) …	
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) …	
Objetivos asociados	• OBJ–*x* <*nombre del objetivo*> …	
Requisitos asociados	• Rx–y <*nombre del requisito*> …	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso { durante la realización de los casos de uso <*lista de casos de uso*>, cuando <*evento de activación*> }	
Precondición	<*precondición del caso de uso*>	
Secuencia normal	**Paso**	**Acción**
	p_1	{El actor <*actor*>, El sistema} <*acción/es realizada/s por actor/sistema*>
	p_2	Se realiza el caso de uso <*caso de uso (RF–x)*>
	p_3	Si <*condición*>, {el actor <*actor*>, el sistema} <*acción/es realizada/s por actor/sistema*>
	p_4	Si <*condición*>, se realiza el caso de uso <*caso de uso (RF–x)*>
	…	…
Postcondición	<*postcondición del caso de uso*>	
Excepciones	**Paso**	**Acción**
	p_i	Si <*condición de excepción*>, {el actor <*actor*>, el sistema} <*acción/es realizada/s por actor/sistema*>, a continuación este caso de uso {continúa, termina}
	p_j	Si <*condición de excepción*>, se realiza el caso de uso <*caso de uso (RF–x)*>, a continuación este caso de uso {continúa, termina}
	…	…
Rendimiento	**Paso**	**Cota de tiempo**
	q	*m* <*unidad de tiempo*>
	…	…
Frecuencia esperada	<*nº de veces*> veces / <*unidad de tiempo*>	
Importancia	<*importancia del requisito*>	
Urgencia	<*urgencia del requisito*>	
Estado	<*estado del requisito*>	
Estabilidad	<*estabilidad del requisito*>	
Comentarios	<*comentarios adicionales sobre el requisito*>	

Figura 3-29 Plantilla y patrones-L para requisitos funcionales (casos de uso)

Otro ejemplo puede se especificar que el usuario puede intentar conectarse al sistema un máximo de tres veces. Una posible especificación sería la que puede verse en la figura 3-30, bastante más *natural* y fácil de entender que la que puede verse en la figura 3-31 utilizando la propuesta descrita en [Coleman 1998].

Secuencia normal	Paso	Acción
	1	El sistema solicita al usuario su nombre de usuario y su clave de acceso
	2	El usuario proporciona el sistema su nombre y su clave de acceso
	3	El sistema comprueba si el nombre de usuario y la clave de acceso son correctas
	4	Si el nombre de usuario y la clave no son correctas, el sistema permite al usuario repetir el intento (pasos 1–3) hasta un máximo de tres veces
	5	Si el nombre de usuario y la clave son correctas, el sistema permite el acceso al usuario
Excepciones	Paso	Acción
	4	Si el usuario ha intentado tres veces acceder sin éxito, el sistema rechaza el acceso del usuario, a continuación este caso de uso termina

Figura 3-30 Ejemplo de caso de uso de conexión de usuario (plantilla)

```
1. El sistema inicializa intentos a 0

2. REPETIR
        2.1 El sistema solicita al usuario su nombre de
        usuario
        2.2 El usuario proporciona al sistema su nombre
        de usuario
        2.3 El sistema solicita al usuario su clave
        2.4 El usuario proporciona al sistema su clave
        2.5  El  sistema  comprueba  si  el
            nombre de usuario y la clave son
            correctas
        2.6 El sistema incrementa intentos
    HASTA QUE el nombre de usuario y la clave sean
    correctas o intentos = 3

3. SI el nombre de usuario y la clave son correctas
      3.1 El sistema permite el acceso al usuario
    SINO
      3.2 El sistema rechaza el acceso del usuario
    FINSI
```

Figura 3-31 Ejemplo de caso de uso de conexión de usuario (Coleman)

Post-condición: en este campo se expresan en lenguaje natural las condiciones que se deben cumplir después de la terminación normal del caso de uso.

Excepciones: este campo especifica el comportamiento del sistema en el caso de que se produzca alguna situación excepcional durante la realización de un paso determinado.

Rendimiento: en este campo puede especificarse el tiempo máximo para cada paso en el que el sistema realice una acción.

Frecuencia esperada: en este campo se indica la frecuencia esperada de realización del caso de uso, que aunque no es realmente un requisito, es una información interesante para los desarrolladores.

3.3.2.2.8.16. Plantilla para requisitos no funcionales

Los requisitos no funcionales del sistema se pueden expresar usando la plantilla que puede verse en la figura 3-32. El único campo específico de esta plantilla es la descripción, en la que se usa un patrón-L que debe completarse con la capacidad que deberá presentar el sistema, el significado del resto de los campos es el mismo que para las plantillas anteriores.

RNF–<*id*>	<*nombre descriptivo*>
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)
Autores	• <*autor de la versión actual*> (<*organización del autor*>) ...
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) ...
Objetivos asociados	• OBJ–*x* <*nombre del objetivo*> ...
Requisitos asociados	• Rx–y <*nombre del requisito*> ...
Descripción	El sistema deberá <*capacidad del sistema*>
Importancia	<*importancia del requisito*>
Urgencia	<*urgencia del requisito*>
Estado	<*estado del requisito*>
Estabilidad	<*estabilidad del requisito*>
Comentarios	<*comentarios adicionales sobre el requisito*>

Figura 3-32 Plantilla y patrones-L para requisitos no funcionales

3.3.2.2.8.17. Plantilla para conflictos

Como ya se ha comentado, durante las sesiones de investigación de requisitos puede ser necesario resolver mediante algún tipo de negociación posibles conflictos en los requisitos revisados en iteraciones previas del proceso. Para documentar dichos conflictos, y las soluciones adoptadas, se propone la plantilla que puede verse en la figura 3-33.

El significado de los campos de la plantilla es el siguiente:

Identificador y nombre descriptivo: al igual que el resto de la información correspondiente a los requisitos, cada conflicto debe poderse identificar de forma única y tener un nombre descriptivo. El prefijo propuesto para lograr una rápida identificación es *CFL*.

Versión, Autores, Fuentes: estos campos tienen el mismo significado que en las plantillas para objetivos y requisitos, aunque referidos al conflicto.

Objetivos y requisitos en conflicto: este campo debe contener una lista con los objetivos y/o requisitos afectados por el conflicto.

Descripción: este campo debe contener la descripción del conflicto.

Alternativas: este campo debe contener una lista con las posibles alternativas de solución que se hayan identificado para solucionar el conflicto.

Solución: este campo debe contener la descripción de la solución negociada del conflicto.

Importancia, Urgencia: estos campos indican respectivamente la importancia y la urgencia de la resolución del conflicto.

CFL–<*id*>	<*nombre descriptivo*>
Versión	<*nº de la versión actual*> (<*fecha de la versión actual*>)
Autores	• <*autor de la versión actual*> (<*organización del autor*>) ...
Fuentes	• <*fuente de la versión actual*> (<*organización de la fuente*>) ...
Objs./Reqs. en conflicto	• OBJ/Ryy–*x* <*nombre del objetivo o requisito en conflicto*> ...
Descripción	<*descripción del conflicto*>
Alternativas	• <*descripción alternativa de solución*> (<*autores alternativa*>) ...
Solución	<*descripción de la solución adoptada (si se ha acordado)*>
Importancia	<*importancia de la resolución del conflicto*>
Urgencia	<*urgencia de la resolución del conflicto*>
Estado	<*estado del resolución del conflicto*>
Comentarios	<*comentarios adicionales sobre el conflicto*>

Figura 3-33 Plantilla para conflictos

Estado: este campo indica el estado de resolución del conflicto, que podrá estar en *negociación* o bien *resuelto*.

Comentarios: este campo tiene el mismo significado que en las plantillas descritas previamente.

3.3.2.2.9. Análisis de requerimientos generados en la investigación de requisitos

Durante el análisis vamos a definir más claramente qué es lo que va a hacer nuestro programa. Debemos hacer varias cosas principalmente:

1. Analizar los casos de uso generados con las plantillas respectivas, que forman parte del Documento de Requisitos de Software.

2. Realizar una validación de los procesos que se representan en los casos de uso junto con el usuario de la nueva aplicación Web a desarrollar.

3. Se deben realizar los ajustes correspondientes a los casos de uso, luego de realizada la validación con los usuarios.

4. Realizar un mini ciclo en la definición de los procesos mediante casos de uso, hasta que las necesidades del usuario se encuentren satisfechas.

3.3.2.2.10. Recomendaciones estructurales para el manejo de la fase "Análisis de Requerimientos"

Se debe realizar luego de tener el documento de requisitos de software el análisis de factibilidad correspondiente a la nueva aplicación Web, para luego realizar las siguientes actividades técnicas. El documento de análisis de factibilidad se debe elaborar en base al "ANEXO-4.3-Documento de

estudio de factibilidad". Las actividades técnicas que se debe realizar en esta fase son:

1. Dar detalle a los casos de uso descritos.
2. Definir una interfaz inicial del sistema (si es aplicable).
3. Desarrollar el modelo del mundo.
4. Validar los modelos.

3.3.2.2.11. Documentos Entregables

Los documentos entregables de la fase "Análisis de requerimientos" son:

1. El formulario de solicitud de requerimientos.
2. El *Documento de Requisitos del software* (DRS) conformado de acuerdo a los estándares de las plantillas:

    2.1. Plantilla y patrones-L para objetivos del sistema.

    2.2. Plantilla y patrones-L para requisitos de almacenamiento de información.

    2.3. Plantilla y patrones-L para actores.

2.4. Plantilla y patrones-L para requisitos funcionales (casos de uso):

Los casos de uso iniciales deben cumplir las siguientes características:

- Requerimientos más importantes del sistema.
- Usuarios y sistemas externos en comunicación.

2.5. Plantilla y patrones-L para requisitos no funcionales.

2.6. Plantilla para conflictos.

3. El formulario de bitácoras de reuniones de acuerdo al estándar de formularios y documentos de la fase.
4. Soporte de entrevistas (apuntes, notas, copias de documentos).
5. Documento de análisis de factibilidad de acuerdo al estándar de formularios y documentos de la fase.

Finalmente para sistemas grandes o complejos se debe elaborar un modelo del mundo inicial preliminar, es decir:

- Diagrama de Clases preliminar, relaciones entre clases y especificación.

3.3.2.3. Modelado conceptual

En esta sección se presenta la forma o manera en cómo desarrollar la fase de modelado conceptual de la metodología de desarrollo propuesta. Se presenta un informe con el detalle de la información de entrada de esta fase así como la salida que ésta genera con la información representada mediante documentos, diagramas, etc.

3.3.2.3.1. Objetivos de la fase

Los objetivos de esta fase son los siguientes:

- Alcanzar un ajuste entre la definición raíz (qué es el sistema) elaborada en la fase de análisis y el modelo conceptual (qué hace el sistema) que por otra parte sean mutuamente consistentes.

- Explicar cuáles son y cómo se relacionan los conceptos o entidades que son relevantes en la descripción del problema.

- Modelar el dominio del problema y la solución mediante el diagrama de clases y de objetos.

- Lograr una descripción esquemática de un sistema, generalmente en la forma de una descripción escrita, o diagrama de flujo, que incluye representaciones visuales correspondientes a propiedades conocidas o inferidas del sistema que está siendo descrito en el modelo.

- Describir el mundo real en términos formales, no ambiguos.

- Representar mediante diagramas de modelado UML los requisitos obtenidos en la fase de análisis.

- Definir objetos del mundo real con sus atributos y relaciones entre ellos.

- Representar mediante un diagrama de clases a los objetos del mundo real.

- Representar las relaciones entre los objetos del mundo real mediante diagramas de comportamiento.

- Captar y almacenar el universo del discurso mediante una descripción rigurosa, representando la información que describe a la organización y que es necesaria para su funcionamiento.

- Aislar la representación de la información de los requisitos y exigencias de cada usuario particular.

- Independizar la definición de la información de los SGBD en concreto.

- Describir los distintos contenidos de información que pueden coexistir en una base de datos.

- Definir los límites de la aplicación identificando aquellos componentes que deberían ser incluidos en la aplicación y aquellos que pueden ser excluidos.

- Elaborar modelos conceptuales que sean los más simples en lo posible, pero teniendo en cuenta que no se debe excluir aquellos componentes cruciales para la solución de nuestro problema.

- Identificar los atributos o unidades de medida de los componentes de la aplicación.

- Clasificar los componentes de la aplicación una vez que se hayan definido los límites de la aplicación, identificando los

componentes que se deben incluir y los que se deben excluir.

- Clasificar los componentes de la aplicación en categorías. Estos componentes se pueden clasificar en al menos siete categorías fundamentalmente diferentes: variables de estado, variables externas, constantes, variables auxiliares, transferencias de material, transferencias de información y fuentes (Forres ter 1961, Innis 1979, Grant 1986).

- Identificar las relaciones entre los componentes de la aplicación que son de interés. Existen dos formas en que los componentes de la aplicación pueden estar relacionados: a través de transferencias de material o mediante transferencias de información.

- Representar formalmente el modelo conceptual mediante el diagrama de clases UML.

- Describir los patrones esperados del comportamiento del modelo acerca del comportamiento de la aplicación, los

cuáles usualmente resultan del mismo conocimiento a priori en que nos basamos para desarrollar el modelo conceptual y de lo aprendido de la aplicación durante el desarrollo del modelo conceptual. Esto se debe realizar mediante los diagramas de interacción propuestos en UML.

- Simplificar la realidad.

- Proporcionar los planos la aplicación.

- Ayudar a visualizar cómo es o cómo queremos que sea la aplicación.

- Especificar la estructura o el comportamiento de la aplicación.

- Proporcionar diagramas que nos guían en la construcción de la aplicación.

- Mediante Diagramas de Secuencia UML representar el comportamiento entre actores y objetos de la aplicación.

.3.2.3.2. Diseño preliminar

Aquí ya empezamos a pensar en cómo vamos a hacer las cosas, es decir, como vamos a desarrollar el diseño de un nuevo sistema.

Viendo los casos de usos, deberíamos ver qué cosas podemos hacer comunes o como librerías aparte, que podamos reutilizar. En este punto y con los casos de uso en general, debemos tener cuidado. Según una crítica generalizada a los casos de uso, estos llevan a un diseño funcional y no a uno orientado a objetos. Debemos tratar de pensar en objetos y almacenarlos juntos en la misma librería cuando estén muy relacionados entre sí, no en funciones. Por ello es buena idea tratar de agrupar las clases del diagrama de clases del negocio en paquetes y tratar de desarrollar la arquitectura a partir de ellas. Es importante en este paso definir las interfaces y relaciones entre paquetes. Para ello puede servir de ayuda hacer los diagramas de secuencia de

los casos de uso mostrando los actores, los paquetes y los mensajes entre ellos.

3.3.2.3.3. Diseño detallado

En el diseño detallado ya se entra a nivel de clases y métodos. Por cada paquete que hayamos extraído en el paso anterior y siguiendo siempre los casos de uso, debemos ir detallando las clases que vamos a implementar y los métodos que van a tener. Detallamos aun más los casos de uso y las interfaces de las clases.

3.3.2.3.4. Recomendaciones estructurales para el manejo de la fase "Modelado Conceptual"

Se deben manejar dos sub-fases para esta fase de la metodología:

- Diseño preliminar del sistema.
- Diseño detallado del sistema.

3.3.2.3.4.1. Diseño preliminar del sistema

En esta sub-fase se define una subdivisión en aplicaciones del sistema (si es lo suficientemente grande) y la forma de comunicación con los sistemas ya existentes con los cuales debe interactuar.

La actividad técnica que se debe realizar en esta sub-fase es:

1. Identificar la arquitectura del sistema.

Los documentos entregables de esta sub-fase son los diagramas de Ejecución, versión inicial:

1. Procesadores.
2. Procesos.
3. Mecanismos de comunicación.
4. Descripción detallada.

3.3.2.3.4.2. Diseño detallado del sistema

En esta sub-fase se adecúa el análisis a las características específicas del ambiente de implementación y se completan las distintas aplicaciones del sistema con los modelos de control, interfaz o comunicaciones, según sea el caso.

Las actividades técnicas de esta sub-fase son las siguientes:

- Agregar detalles de implementación al modelo del mundo.
- Desarrollar el modelo de interfaz.
- Desarrollar los modelos de control, persistencia y comunicaciones.

Finalmente se deben desarrollar los modelos de control, persistencia y comunicaciones.

3.3.2.3.5. Documentos Entregables

Los documentos entregables de la fase "Modelado Conceptual" son:

- Diagramas de clases y paquetes, con el detalle de la implementación.
- Diagramas de interacción con el detalle de las operaciones más importantes del sistema.
- Diagramas de estados y/o actividades para las clases concurrentes o complejas.
- Diagrama Entidad/Relación de base de datos.

3.3.2.4. Elaboración de componentes

3.3.2.4.1. Objetivos de la fase

Los objetivos de esta fase son los siguientes:

- Crear componentes reusables y reemplazables.

- Garantizar la correcta interacción de todos los componentes del sistema.

- Crear componentes con las características del paradigma orientado a objetos: unificación de datos y comportamiento, identidad y encapsulamiento.

- Permitir que cada aplicación cliente de un componente dependa exclusivamente de la especificación del componente y no de su implementación mediante el uso de interfaces.

- Completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar el cambio de los artefactos construidos, ejecutar el plan de administración de recursos y mejoras en el proceso de desarrollo para el proyecto.

- Prepara la documentación para la instalación o despliegue.

- Identificar a partir de las clases generadas en las actividades anteriores el conjunto de interfaces y especificaciones de componentes que poblarán la arquitectura.

- Crear un conjunto inicial de interfaces y especificaciones de componentes, tanto a nivel de componentes de la aplicación como de componentes de negocio.

- Producir el modelo de tipos del negocio inicial, partiendo del modelo conceptual preliminar.

- Presentar las interfaces y especificaciones de componentes en una arquitectura de componentes inicial, decidiendo de qué forma se agrupan las interfaces en especificaciones de componentes.

- Refinar las definiciones de las interfaces de sistema.

- Definir las interacciones entre los componentes identificando operaciones en las interfaces de los componentes de

negocio y determinando las dependencias entre componentes.

- Definir políticas de manejo de integridad referencial.

- Una vez que la arquitectura e interfaces de los componentes estén estables se debe revisar el nivel de reemplazabilidad de los componentes para hacer posible el re-uso de componentes en futuros proyectos.

- Definir el modelo de información de cada interfaz; este modelo representa una vista abstracta de la información manejada por el componente.

- Especificar formalmente las operaciones de las interfaces; esta especificación se realiza con contratos de software.

- Capturar y documentar las restricciones entre los componentes.

3.2.4.2. Explotando las características de Oracle ADF Framework

El entorno administrado de Oracle ADF Framework permite a los programadores mejorar el modelo de programación para hacerlo compatible con una amplia gama de funcionalidades. Las instrucciones de diseño de Oracle ADF Framework tienen como finalidad fomentar la coherencia y la previsibilidad en las API públicas al habilitar la integración entre lenguajes y el Web. El diseño incoherente de componentes influye de un modo desfavorable en la productividad de los programadores.

"Oracle Application Development Framework" (Oracle ADF) simplifica muchas de las tareas de desarrollo de aplicaciones J2EE haciendo innecesario la implementación de patrones o codificación de tareas repetitivas. Oracle ADF implementa un conjunto de patrones de diseño que pueden ser reutilizados.

El personal técnico de la SBS recibió a fines del año 2007 el curso de Oracle ADF Framework en donde se observaron las

mejores prácticas de este marco de trabajo o framework. Es por esta razón que los conocimientos adquiridos sobre Oracle ADF, se unen a la forma de realizar la fase de "elaboración de componentes" de esta metodología. Lo que sí se recomienda es revisar la parte de documentos entregables de esta fase y la matriz de elementos estándares por cada fase.

.3.2.4.3. Recomendaciones estructurales para el manejo de la fase "Elaboración de componentes"

En esta etapa se desarrolla el código de una manera certificada. Para conseguir esto se recomiendan realizar las siguientes actividades técnicas. Las actividades técnicas que se deben desarrollar en esta fase son las siguientes:

- Utilizar los estándares de programación definidos en el "ANEXO-5-Estándares de codificación-Java".
- Codificación con pruebas unitarias.
- Pruebas de módulos y de sistema integrales luego de terminar su respectivo desarrollo.

3.3.2.4.4. Documentos Entregables

Los documentos entregables de la fase "elaboración de componentes" son las siguientes:

- Código fuente estructurado de acuerdo a los estándares propuestos en el "ANEXO-5-Estándares de codificación-Java".

- Documento de soporte de pruebas unitarias (casos de prueba).

- Documentación del código (generar javadoc).

- Documento de instalación y despliegue de acuerdo a los estándares propuestos en el "ANEXO-8-Estándares de instalación y despliegue".

- Manual de usuario de la aplicación Web según los estándares de presentación del "ANEXO-9-Guía para realizar el manual de usuario de las aplicaciones".

- Manual técnico de la aplicación Web según los estándares de presentación del "ANEXO-10-Guía para realizar el manual técnico de las aplicaciones".

3.3.2.5. Pruebas e implementación

De esta fase se puede recomendar que se deban escribir los "casos de prueba". Básicamente son como la descripción de los casos de uso, pero indicando datos concretos que el operador va a introducir y qué resultados exactos debe dar nuestro programa.

3.3.2.5.1. Objetivos de la fase

Los objetivos de esta fase son los siguientes:

- Observar si la aplicación cumple los requisitos del análisis.

- Ver si la aplicación cumple las especificaciones de diseño.

- Asegurar que el software esté disponible para los usuarios finales.

- Ajustar los errores y defectos encontrados.

- Capacitar a los usuarios y proveer el soporte técnico necesario.

- Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto al inicio del mismo.

- Encontrar defectos en el software. Una prueba tiene éxito si descubre un defecto. Una prueba fracasa si hay defectos pero no los descubre

- Probar el programa completo.

- Generar uno o varios casos de prueba por cada requisito o caso de uso especificado.

- Asegurar la calidad de la aplicación Web

- Revisar posibles cambios en el soporte de las tareas de instalación o despliegue.

3.3.2.5.2. Pruebas tempranas del sistema/software

Una tarea vital en el desarrollo de software es probar la correcta implementación de los requisitos funcionales. Sin embargo, muchas veces la fase de pruebas del sistema es demasiado corta para diseñar un buen conjunto de pruebas, ejecutarlas y analizar sus resultados. Una solución es la estrategia de pruebas tempranas, la cual consiste en obtener de manera sistemática casos de prueba en etapas tempranas del desarrollo. Esta metodología describe los puntos clave de una nueva propuesta de pruebas tempranas del sistema, la cual resuelve algunas

carencias detectadas en estudios anteriores sobre la fase de pruebas del desarrollo de software.

El incremento de la complejidad de los sistemas o software incrementa a su vez la necesidad de asegurar su calidad. La fase de prueba del sistema ayuda a asegurar la calidad del software. La mayoría de las pruebas en la industria se desarrolla a nivel del sistema. Sin embargo, muchas técnicas de prueba del sistema están descritas sólo de manera informal **[47]**. Además, la fase de prueba del sistema suele realizarse al final del proceso de desarrollo, por lo que estas pruebas suelen realizarse de manera superficial e incompleta **[47]**.

En esta metodología, se define un caso de prueba como un reemplazo de un actor del sistema. El caso de prueba simula las interacciones del actor con el sistema, para verificar que el sistema hace lo que se espera de él. Así, el principal artefacto para obtener pruebas del sistema son los requisitos funcionales.

La prueba temprana que propone esta metodología de desarrollo, consiste en la generación de casos de prueba en etapas tempranas del desarrollo de software en paralelo con el desarrollo del mismo. Esta no es una idea nueva, como se verá más adelante. Sin embargo, dos estudios **[41]** y **[43]** (con 22 propuestas en total) muestran que existen muchas carencias en las propuestas existentes. Las propuestas existentes no son completas. Esto significa que describen cómo obtener resultados parciales, principalmente objetivos de prueba, sin describir otros elementos importantes como datos de prueba, resultados esperados o pruebas ejecutables.

En propuestas existentes, se muestra cómo generar casos de prueba para una aplicación Web **[44]**. Esta metodología presenta una nueva propuesta original que intenta resolver las carencias encontradas en las propuestas mencionadas. La nueva propuesta permite generar pruebas ejecutables a partir de casos de uso definidos en prosa y ha sido diseñada específicamente para ser aplicada en etapas tempranas, cuando el sistema aún

no está construido o, incluso, diseñado. Además la propuesta de esta metodología se apoya en el perfil de pruebas de UML (UMLTP) siempre que sea posible **[46]**.

3.3.2.5.3. Caso práctico

El sistema bajo prueba SPJ (Sistema de Providencias Judiciales) es un software para administrar las providencias judiciales electrónicas de la SBS. El caso de uso elegido es "Cargar documento". Este caso de uso se define en la tabla V utilizando la plantilla para requisitos funcionales propuesta en la sección de técnicas de levantamiento e investigación de requisitos.

3.3.2.5.4. Generación de casos de prueba a partir de los casos de uso

3.3.2.5.4.1. Actividades para la generación de casos de prueba

Se presenta en esta metodología un proceso de seis actividades para la fase de generación de pruebas del sistema

a partir de casos de uso. Dichas actividades se muestran en la figura 3-34 y se describen en los siguientes párrafos.

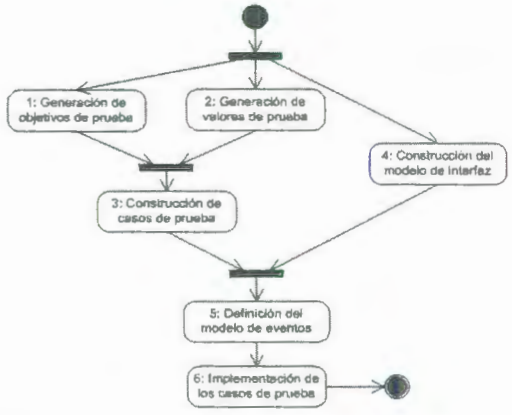


Figura 3-34 Actividades para la generación de pruebas tempranas.

Descripción	Cargar documento
Precondición	No
Escenario principal	1. El usuario selecciona la opción de cargar documento 2. El sistema solicita el documento a cargar. 3. El usuario selecciona el fichero que contiene el documento. 4. El sistema carga el documento. 5. El sistema muestra el documento.
Alternativas / Errores	3. En cualquier momento el usuario puede cancelar la operación y este caso de uso termina. 4. Si el fichero no existe o hay un error, el sistema muestra un mensaje y este caso de uso termina.
Post-condiciones	No

Tabla V. Descripción de caso de uso Cargar documento

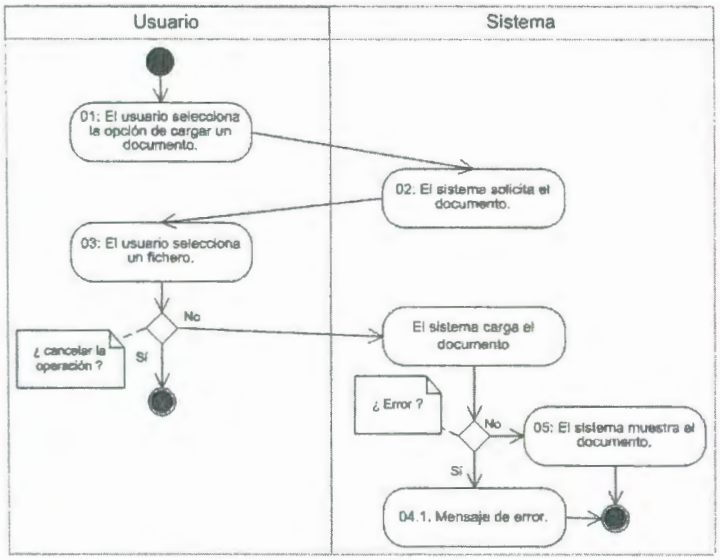
La primera actividad consiste en construir un modelo de comportamiento y obtener objetivos de prueba a partir de él. A continuación, en la segunda actividad, se identifican las

variables del caso de uso **[42]** y se definen los datos de prueba. En la tercera actividad se generan casos de prueba combinando los objetivos de prueba con los valores de prueba. A continuación, en la cuarta actividad se construye el modelo de interfaz. En la quinta actividad, los objetivos de prueba se refinan y se construyen los árbitros que comprobarán si el resultado del caso de prueba es el esperado **[46]**. Finalmente, toda la información generada se implementa en pruebas ejecutables y test harness **[42]**. A continuación se definen con mayor detalle las actividades y modelos.

3.3.2.5.4.2. Generación de objetivos de prueba

El primer paso es la construcción del modelo de comportamiento. El objetivo del modelo de comportamiento es expresar la información contenida en una plantilla de caso de uso de una forma manipulable sistemáticamente. Para efectos de implementación de esta metodología se han seleccionado los diagramas de actividades. El modelo de

comportamiento correspondiente al caso de uso de la tabla V se muestra en la figura 3-35.



Id	Objetivo de prueba
1	01, 02, 03, 04, 04.1
2	01, 02, 03, 04, 04.2
3	01, 02, 03

Figura 3-35 Modelo de comportamiento y objetivos de prueba

En la figura anterior se muestran todos los posibles caminos obtenidos con el criterio de todas las actividades y todas las transiciones.

3.3.2.5.4.3. Generación de valores de prueba

Como se ha mencionado, en primer lugar se identifican las variables **[42]**. El modelo de comportamiento de la figura

3-35 revela dos variables. La primera, llamada OpcionUsuario, indica si el usuario selecciona la opción de cargar el fichero o cancelar la operación. La segunda, llamada Fichero, indica el fichero elegido por el usuario. Ambas variables se definen en la figura 3-36a y 3-36b.

A continuación, el dominio de las variables se divide en categorías **[42]**, **[45]**. En la figura 3-36a y 3-36b se muestra una posible partición utilizando la notación propuesta por el UMLTP. Finalmente, para cada partición que se desee probar se selecciona al menos un valor de prueba. Los calores de prueba del caso práctico se muestran en el diagrama de objetos de la figura 3-36c.

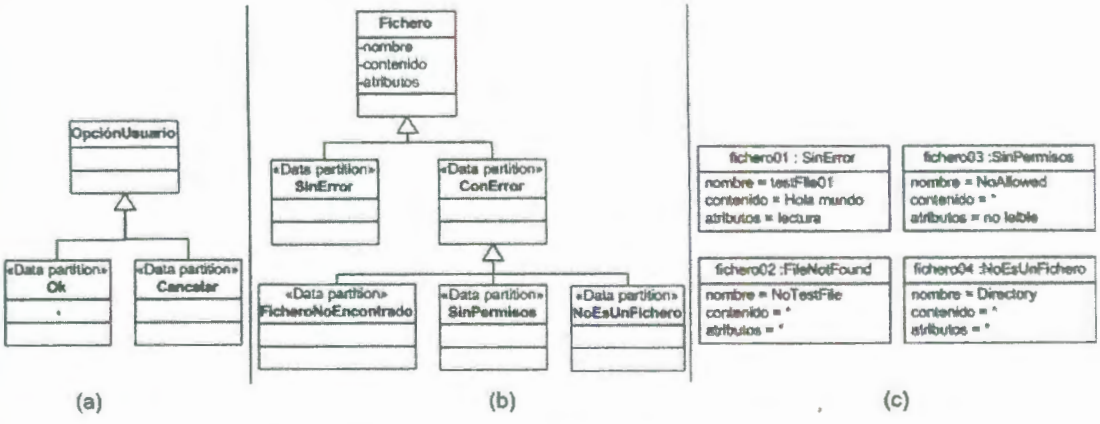


Figura 3-36 Datos de prueba

3.3.2.5.4.4. Construcción de los casos de prueba

En esta actividad se combina los objetivos y los valores de prueba. A partir de tres objetivos y dos variables se han obtenido un total de 5 casos de prueba. Todas las combinaciones se muestran en la tabla VI.

Id	Objetivo	Valor de prueba
1	1	OpcionUsuario = Ok, Fichero = fichero01
2	2	OpcionUsuario = Ok, Fichero = fichero02
3	2	OpcionUsuario = Ok, Fichero = fichero03
4	2	OpcionUsuario = Ok, Fichero = fichero04
5	3	OpcionUsuario = Cancelar, Fichero = *

Tabla VI. Combinación de objetivos y valores de prueba

3.3.2.5.4.5. Construcción del modelo de interfaz de usuario

Al ser pruebas tempranas se asume en este ejemplo que el SPJ (Sistema de Providencias Judiciales) aún no se ha construido y sus interfaces gráficas aún no son las definitivas (o ni siquiera se han diseñado).

Para poder refinar los objetivos de prueba se construirá una interfaz genérica con todos los elementos necesarios para que el caso de prueba pueda interactuar con el sistema. Para ello, usaremos el modelo de componentes de la figura 3-37a. Este modelo es fácilmente extensible. A partir de la definición del caso de uso, se identifican las pantallas y los componentes que el sistema debe ofrecer para llevarlo a cabo. El modelo de interfaz para el caso de uso descrito en la tabla V se muestra en la figura 3-37b.

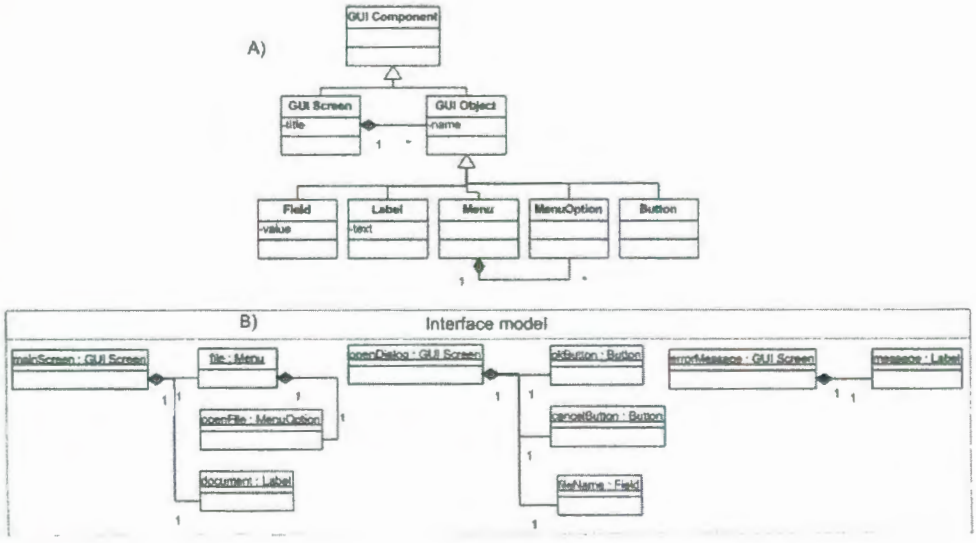


Figura 3-37 Modelo de componentes e interfaz del sistema

3.3.2.5.4.6. Definición del modelo de eventos

Una vez que se dispone de la interfaz del sistema, aunque definida de una manera genérica, es posible refinar los objetivos. Para refinar las actividades realizadas por la prueba sobre el sistema propongo un sencillo conjunto de instrucciones que se recogen en la tabla VII. A partir de las actividades realizadas por el sistema se determinan los resultados esperados y se definen los árbitros **[46]** que verificarán si la prueba se superó o no. Para definir los árbitros proponemos un sencillo conjunto de instrucciones que se recogen también en la tabla VII.

Instrucción	Descripción
ClickOn(**component**)	Representa una pulsación con el botón izquierdo sobre el componente indicado.
SetField(**field, value**)	Asigna al campo el valor indicado.
Assert(**component attribute, value**)	Verifica que el atributo del componente indicado coincide con el valor.
Screen(**GUIScreen**)	Verifica que la pantalla que muestra el sistema coincide con la pantalla indicada.

Tabla VII. Instrucciones para expresar la interacción del usuario con el sistema

La figura 3-38 muestra el modelo de secuencia y los árbitros del primer objetivo de prueba según la notación del UMLTP.

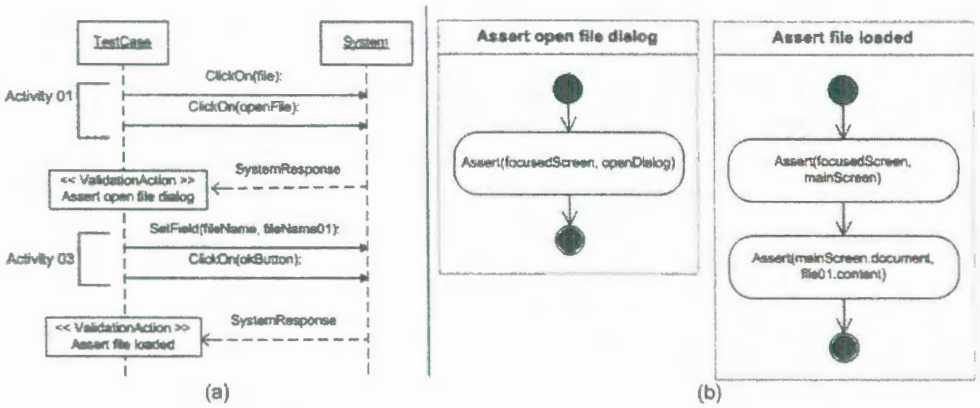


Figura 3-38 Caso de prueba para el primer objetivo

El proceso de prueba temprana ha terminado. Se han generado acciones de prueba, valores de prueba y los resultados esperados. La última actividad consiste en traducir todo lo esperado a pruebas ejecutables. Existen numerosas herramientas para definir pruebas. En nuestro caso práctico hemos elegido la aplicación Stylepad que acompaña al kit de desarrollo de Java y la herramienta Abbot (abbot.sourceforge.net).

Esta propuesta sobre la realizar pruebas tempranas al sistema o software que se esté desarrollando, ha mostrado un proceso para la prueba temprana del sistema a partir de

los casos de uso del SPJ (Sistema de Providencias Judiciales). Aunque este proceso se ha desarrollado para probar la funcionalidad desde la perspectiva de actores humanos, puede también usarse para probar otros tipos de actores. De hecho, en el caso de estudio todas las pruebas han sido diseñadas antes de elegir una implementación real.

3.3.2.5.5. Diagramas de componentes y diagramas de despliegue para la implementación

Los diagramas que se deben realizar en la fase "pruebas e implementación" son los de componentes y despliegue. Una breve descripción de estos diagramas es la que se presenta a continuación.

3.3.2.5.6. Recomendaciones estructurales para el manejo de la fase "Pruebas e implementación"

Para esta fase se deben realizar las siguientes actividades:

- Generar los Casos de prueba basados en los casos de uso originales de la fase "análisis de requerimientos" mediante las recomendaciones de la metodología.

- Ejecutar el procedimiento de instalación y despliegue de los módulos para luego corregir errores de codificación y de rendimiento que se encuentra en el "ANEXO-8-Estándares de instalación y despliegue".

3.3.2.5.7. Documentos Entregables

Los documentos entregables de la fase "pruebas e implementación" son las siguientes:

- Documentos de soporte de pruebas unitarias (casos de uso con datos de prueba) de todos los módulos del sistema (aplicación Web).

- Si la aplicación Web lo requiere se debe hacer un seguimiento en la ejecución de la instalación o despliegue, en el área de administración de recursos tecnológicos de la SBS.

3.3.3. Flujo de las fases

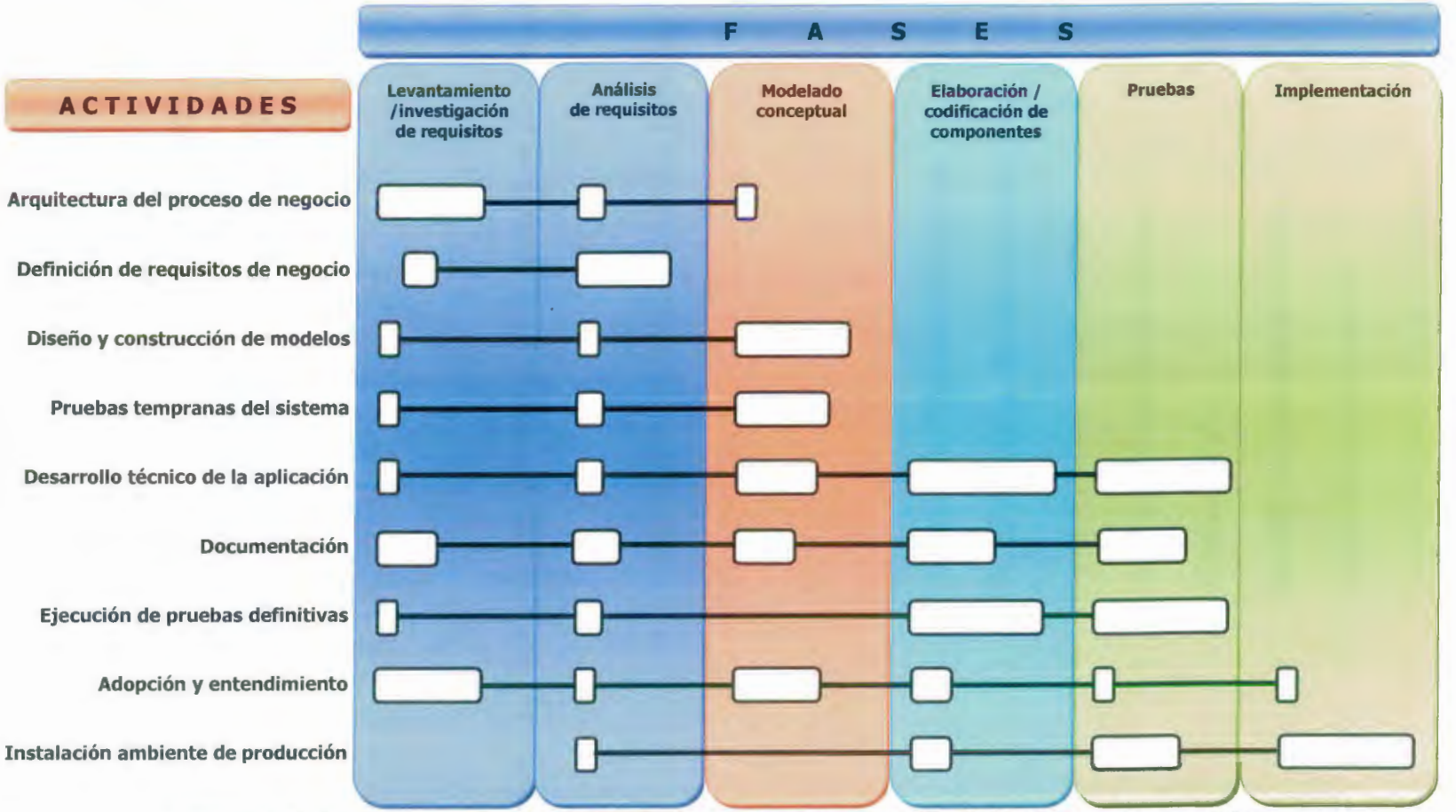


Figura 3-39. Diagrama de flujo de las fases de la metodología de desarrollo

Esta metodología de desarrollo tiene las siguientes recomendaciones sobre las actividades generales que se deben realizar en cada una de sus fases o sub-fase. Estas actividades definidas para cada fase o sub-fase, se pueden realizar en secuencia o paralelo dependiendo de las características de cada proyecto.

3.3.3.1. Actividades generales para cada fase de la metodología

Las actividades que se deben realizar por cada fase o sub-fase de la metodología de desarrollo son las siguientes:

1. Arquitectura del proceso de negocio.
2. Definición de requisitos de negocio.
3. Diseño y construcción de modelos.
4. Pruebas tempranas del sistema.
5. Desarrollo técnico de la aplicación.
6. Documentación.
7. Ejecución de pruebas definitivas.
8. Adopción y entendimiento.
9. Instalación ambiente de producción.

3.3.3.1.1. Arquitectura del proceso de negocio

El objetivo de establecer una arquitectura del proceso de negocio de la SBS es proveer un marco de trabajo para combinar los cambios en los procesos de negocios de las unidades de la SBS con la implementación de las aplicaciones Web.

Por ahora la SBS completará de forma manual esta actividad por medio de la Subdirección de Desarrollo Institucional de la SBS, debido a que el documento de los procesos de todas las áreas o unidades de la SBS se encuentran en estado de reingeniería, y por otro lado se encuentra la reciente adquisición de los productos Oracle relacionados a la arquitectura SOA por parte de la SBS. Esto conlleva a que los cursos de capacitación están pendientes de ejecutarse a finales del 2008.

Entonces esta actividad consiste en agregar al documento de análisis de factibilidad el vínculo al proceso interno manual de las unidades de la SBS, al que se desea dar soporte en su progresiva automatización, y de ser el caso también contemplar una automatización completa.

3.3.3.1.2. Definición de requisitos de negocio

Esta actividad trata lo relacionado con la interpretación, aclaración y entendimiento de los requerimientos del cliente. Siempre se debe definir primero los procesos inmersos en el requerimiento y sobre ellos realizar el planteamiento del programa. Nunca se debe definir los procesos a partir del análisis de requerimientos. Es un error muy común entender que los desarrolladores de software son analistas de procesos.

El análisis de requerimientos genera el documento de especificación de requerimientos del software: que es la descripción completa del comportamiento y la lista de

funcionalidades del programa a desarrollar, como se explico en secciones anteriores.

3.3.3.1.3. Diseño y construcción de modelos

Mediante la definición de la arquitectura del software se especifica su estructura que comprende: los componentes, las propiedades externas de estos componentes y la relación entre ellos.

En esta actividad también se debe definir la plataforma sobre la cual funcionara el software. La plataforma hace referencia al hardware y software usado tanto para el desarrollo como para la implantación.

Se debe definir el ambiente tanto de desarrollo como de implantación: hardware, computadores, impresoras, sistema

operativo, lenguaje de desarrollo, base de datos, drivers, dispositivos, etc.

Esta actividad contempla el diseño del modelo de clases y del modelo físico de base de datos (diagrama E/R) del software a desarrollar.

Mediante el diseño se deben especificar la declaración de las clases y operaciones de cada funcionalidad. En lo posible se recomienda apoyarse en UML para la especificación de cada clase y generar el diagrama de clases.

El diagrama E/R es el bosquejo básico del diseño de base de datos pero el entregable de esta sub-actividad es el *script* de generación o modificación de la estructura de datos.

Se debe especificar las tablas, campos y relaciones y definir los índices, llaves primarias, llaves foráneas, reglas de integridad, datos por defecto, etc.

El diseño de la interfaz de usuario también forma parte de esta actividad, y se recomienda que siempre sea validada por el usuario final. La interfaz del software es la parte visual del programa y la que más aprecia el usuario final. El usuario final quiere algo agradable, amigable, fácil de usar, completo pero sencillo, estándar y personalizable. Los programas deben tener un diseño homogéneo.

Se debe tener en cuenta el tamaño de las ventanas, de la fuente, los colores, los fondos, la usabilidad, la ergonomía, etc. También hacer parte de la interfaz los reportes, los archivos de salida y en general cualquier exposición del programa a nuestros sentidos.

3.3.3.1.4. Pruebas tempranas del sistema

Una tarea vital en el desarrollo de software es probar la correcta implementación de los requisitos funcionales. Sin embargo, muchas veces la fase de pruebas del sistema es demasiado corta para diseñar un buen conjunto de pruebas, ejecutarlas y analizar sus resultados, por ejemplo en el caso de pedidos de la autoridad máxima de la SBS. En esta metodología se describe una propuesta válida de pruebas tempranas sobre los requisitos funcionales de la solución informática a desarrollar en el caso de aplicaciones Web. Para mayor referencia de cómo elaborar los casos de prueba, revisar la sección en donde se detalla la fase de "Pruebas e implementación" de la metodología de desarrollo.

3.3.3.1.5. Desarrollo técnico de la aplicación

Esta actividad comprende a la codificación de las clases y sus operaciones. La codificación incluye: interpretación, creación, reutilización, refinamiento, integración, compilación y prueba del código fuente. También hace parte de esta actividad el control

de errores, manejo de transacciones, empaquetamiento de clases, comunicación entre desarrolladores, integración de código fuente, etc.

Las tareas recomendadas son las de: integración y entrega. La integración consiste en tomar los fuentes de la entrega anterior y agregar el código fuente de las nuevas funcionalidades desarrolladas y generar el programa ejecutable y/o el instalador según sea conveniente. Se entrega el programa, los *scripts* de cambios en la estructura de datos, la lista de cambios de todos los componentes y los nuevos componentes se que usaron para el desarrollo de la entrega.

3.3.3.1.6. Documentación

En la práctica toda aplicación debe cumplir con un marco regulado por documentos (leyes, decretos, normas, políticas, circulares, principios, etc). Es común entregar a los desarrolladores los documentos que regulan el programa

cuando lo correcto es que se les entregue los procesos. El análisis documental ayuda a entender y precisar los requerimientos. En esta actividad se debe definir un cronograma de trabajo para todos los proyectos de desarrollo de software.

El costo de un proyecto se fundamenta en los tiempos que toma el desarrollo de cada actividad. La duración de cada actividad se debe estimar en horas hombre. Siempre hay que desconfiar de los cronogramas con tiempos muy cortos. Y se debe alejar del análisis de tiempos las presiones y premuras que provienen del solicitante, usuarios finales, administrador del proyecto y la misma SBS.

Se recomienda que los días laborales no sobrepasen la carga normal de horas de trabajo. Las largas jornadas laborales no siempre llevan a acelerar las tareas. Es mejor tener un cronograma largo y viable que corto e ilusorio. En el cronograma también se debe agregar las tareas ya realizadas.

Se deben actualizar los manuales técnicos y manuales de usuario, de tal manera que no se acumule la tarea de redacción de éstos al final del desarrollo.

Tan pronto se tiene lista cada funcionalidad se debe actualizar los manuales: el técnico y el de usuario final. Se sugiere que el manual de usuario esté incluido en el programa y que sea contextual, es decir, que el usuario en cualquier momento tenga la opción de solicitar la ayuda en línea de la funcionalidad que esté trabajando.

3.3.3.1.7. Ejecución de pruebas definitivas

Las pruebas del desarrollo se deben realizar en un ambiente más parecido al ambiente de producción que al de pruebas. En lo posible usar una base de datos que no sea la de desarrollo. Se deben realizar pruebas con datos lo más parecido a la realidad.

Cuidado con las pruebas donde se llenan todos los campos con "1" o con mezclas de caracteres no descifrables. Las pruebas se basan en el formulario de casos de uso de pruebas establecidos para cada funcionalidad. Un buen examen e inspección de lo desarrollado genera tranquilidad y sobre todo apoya a la calidad del software.

3.3.3.1.8. Adopción y entendimiento

Se deben elaborar planes de entrega de los módulos del software que se esté desarrollando. El plan de entregas es la planificación del desarrollo de las funcionalidades por grupos o módulos con el fin de entregar un programa totalmente funcional y listo para entrar a producción.

Para cada entrega se debe planificar las funcionalidades a desarrollar y el periodo de tiempo que tomará su desarrollo. El

periodo entre entregas puede ser de 1 mes siempre que el proyecto tome más de 2 meses, para proyectos pequeños las entregas pueden ser cada 15 días.

El periodo es propio de cada proyecto, de cada grupo de desarrollo y de cada empresa; posiblemente generar programas cada cierto tiempo funciones correctamente para unas empresas y no para otras, todo depende de la dinámica del procedo de desarrollo: personal exclusivo para desarrollo, suficiente personal de pruebas, requerimientos cambiantes, calidad de los desarrolladores, etc.

3.3.3.1.9. Instalación ambiente de producción

Se debe preparar un ambiente de pruebas lo más similar al ambiente de producción y que no sea el mismo que el ambiente de pruebas de desarrollo. Se ejecuta la actualización sugerida por el desarrollador y se pasa al usuario líder para sus pruebas.

Se debe realizar como parte de esta actividad las respectivas capacitaciones para el uso del software.

Se capacita al usuario líder respecto al nuevo desarrollo. Esta capacitación no se debe realizar en un lenguaje técnico sino basado en la terminología del proceso que se apoya.

Luego el usuario líder realiza las pruebas de cada funcionalidad de acuerdo con el formulario de casos de uso, redactado de acuerdo a la lista de funcionalidades. Para las pruebas se debe tener en cuenta además los requerimientos no funcionales. En tanto que se desarrolla las pruebas el desarrollador inicia el desarrollo de la próxima entrega.

De acuerdo con las pruebas, el usuario líder determina la aceptación del paso a producción del programa con sus

funcionalidades. La no aceptación implica un requerimiento de corrección al desarrollador y la realización de nuevas pruebas.

Luego se deben tomar en cuenta las siguientes tareas: Adecuación de la plataforma, paso a producción del programa y capacitación de usuarios.

La capacitación al usuario final se debe realizar con anticipación al paso a producción para que los usuarios conozcan los cambios y se preparen ante las innovaciones tecnológicas. El paso a producción implica la adecuación y actualización de la plataforma y la distribución y/o instalación del programa.

3.4. Estándares de cada fase de la metodología de desarrollo

En esta sección se describe un resumen de los recursos utilizados para llevar a cabo cada fase y/o sub-fase de la metodología de desarrollo.

Entre los recursos utilizados se consideran a los diagramas, documentos y formularios. Para cada fase de la metodología se utilizan los siguientes recursos:

- **Análisis de requerimientos:** para esta fase se utiliza el formulario de solicitud de requerimientos, el documento de requisitos del Software, el formulario de bitácoras de reuniones, el documento de análisis de factibilidad, diagrama de casos de uso UML, y el diagrama de clases preliminar UML.

- **Modelado conceptual:** en esta fase se deben utilizar los diagramas de clases y paquetes UML, los diagramas de interacción y los diagramas de estados y/o actividades UML.

- **Elaboración / codificación de componentes:** para esta fase se deben utilizar los estándares de codificación en lenguaje Java, casos de prueba para los programas, los estándares de diseño visual de pantallas, estándares de instalación y despliegue para

generar manuales, y los estándares para realizar los manuales de usuario y manuales técnicos de las aplicaciones.

- **Pruebas e Implementación:** en esta fase se deben utilizar casos de prueba integrados de toda la aplicación Web, los diagramas de componentes y los diagramas de despliegue UML.

Nota: el documento de estándares de nomenclaturas y abreviaturas se debe utilizar en todas las fases de la metodología de desarrollo. El detalle de este documento se encuentra en el "ANEXO-7-Estándares de nomenclaturas y abreviaturas".

.4.1. Documentos y formularios

En esta sección se presenta los esquemas de los documentos, diagramas y formularios que se recomienda utilizar en cada fase de la metodología de desarrollo.

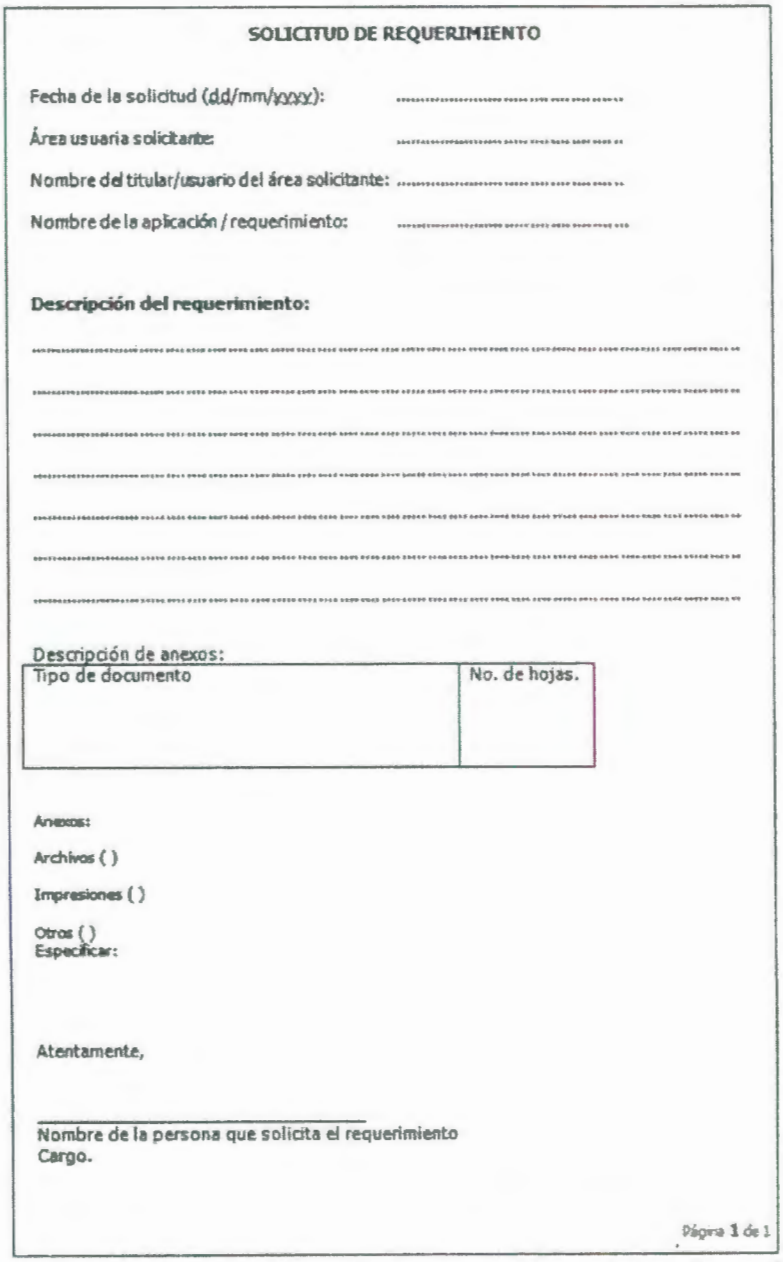
3.4.1.1. Formulario para la solicitud de requerimientos

Este formulario se debe utilizar al inicio del proceso de desarrollo de una nueva aplicación Web. En la figura 3-40 se describe el formulario que deben utilizar los usuarios de las unidades administrativas de la SBS, para solicitar el desarrollo de un nuevo software. Este formulario debe iniciar el proceso normal de un nuevo desarrollo de software, y debe estar dirigido a la Dirección Nacional de Recursos Tecnológicos. Este formulario contiene los siguientes campos:

Fecha de la solicitud: Es la fecha de la solicitud de desarrollo de un nuevo producto de software.

Área usuaria solicitante: Este campo se refiere a la descripción de la unidad administrativa de la SBS que solicita el nuevo desarrollo.

Nombre del titular/usuario del área solicitante: Sirve para indicar el nombre del responsable del área solicitante del nuevo desarrollo o para registrar el nombre de la persona delegada por cada área para realizar la solicitud de desarrollo de software.



SOLICITUD DE REQUERIMIENTO

Fecha de la solicitud (dd/mm/yyyy): .....

Área usuaria solicitante: .....

Nombre del titular/usuario del área solicitante: .....

Nombre de la aplicación / requerimiento: .....

Descripción del requerimiento:

.....

.....

.....

.....

.....

.....

.....

Descripción de anexos:

Tipo de documento	No. de hojas.

Anexos:

Archivos ( )

Impresiones ( )

Otros ( )
Especificar:

Atentamente,

_____
Nombre de la persona que solicita el requerimiento
Cargo.

Página 1 de 1

Figura 3-40 Formulario para la solicitud de desarrollo de software

Nombre de la aplicación/requerimiento: En este campo se debe ingresar el nombre de la aplicación o del requerimiento del nuevo desarrollo.

Descripción del requerimiento: Aquí se debe especificar un resumen del requerimiento del desarrollo del nuevo software, sin llegar a detalles específicos de funcionalidad del software.

Descripción de anexos: Esta sección del formulario de desarrollo contiene dos columnas para hacer referencia a los archivos anexos a la solicitud de desarrollo. La primera columna se refiere al tipo de documento del archivo anexo. La segunda columna sirve para indicar el número de hojas que tiene el archivo anexo.

Anexos: Existe luego de la sección de los archivos anexos, una parte para indicar archivos anexos especiales, como por ejemplo: archivos de correos electrónicos impresos o digitales, o cualquier

otro tipo especial de archivos anexos que se necesite adjuntar a la solicitud del nuevo desarrollo de software.

Firma del solicitante: En este campo se debe registrar la firma de la persona responsable de la solicitud del nuevo desarrollo, junto al cargo que desempeña en la SBS.

El formato para la impresión del formulario para la solicitud de requerimientos se encuentra en el "ANEXO-4.1-Formulario para la solicitud de requerimientos".

3.4.1.2. Formulario de bitácoras de reuniones con el usuario

Este formulario se debe utilizar como soporte a las entrevistas sostenidas con el usuario una vez que se ha iniciado el proceso de desarrollo. El formulario de bitácoras de reuniones se debe utilizar en la fase de análisis de requerimientos y se puede utilizar en las

fases de modelado conceptual, elaboración de componentes y pruebas e implementación.

En la figura 3-41 se describe el formulario que deben utilizar los ingenieros de software de la SBS, para documentar el desarrollo de entrevistas con el usuario y puede ir como anexo al documento principal de la fase de análisis de requerimientos "Documento de Requisitos de Software".

Este formulario contiene los siguientes campos:

Lugar: sirve para especificar la ubicación de donde se lleva a cabo la reunión con el usuario como por ejemplo: sala de reuniones de la unidad de la SBS.

Fecha: se debe indicar la fecha en la que se suscita la reunión en formato dd/mm/yyyy.

Horario: este campo es para registrar el horario de inicio y de fin de la reunión en formato hh:mm:ss.

Proyecto: sirve para registrar la descripción del proyecto de desarrollo para una aplicación Web específica.

Etapa: en este campo se debe especificar la descripción de la fase o etapa del proceso de desarrollo.

Tema a tratar: sirve para describir el asunto principal de la reunión.

Asistentes: en este campo se debe indicar el nombre de las personas que participan en la reunión.

Asuntos tratados en la reunión: en este campo se debe especificar un resumen sobre los puntos tratados en la reunión y de las limitaciones o restricciones al momento de encontrar alguna solución a un problema.

Pendientes: sirve para registrar los asuntos pendientes de resolver o tratar para una próxima reunión.

Conclusiones: sirve para especificar las conclusiones con respecto a cada asunto tratado en la reunión.

Próxima reunión: si se acuerda en la reunión hacer otra reunión, entonces se debe especificar la fecha de la próxima reunión en formato dd/mm/yyyy.

Acuerdos alcanzados: sirve para especificar una lista de acuerdos entre los participantes de la reunión.

BITACORA DE REUNION

Bitácora de reuniones en el proceso de desarrollo de aplicaciones de la Superintendencia de Bancos y Seguros, Dirección Nacional de Recursos Tecnológicos, Subdirección de Desarrollo y Aplicaciones Tecnológicas.

Lugar: (Lugar donde se llevo a cabo la reunión).....

Fecha: (Fecha de la reunión).....

Horario: Duración de la reunión, hora (mm:ss) de inicio y fin.....

Proyecto: (Nombre del proyecto al cual se esta haciendo referencia)

Etapa: (Etapa dentro del proceso de desarrollo en la que tiene lugar esta reunión)

Tema a tratar: .....

Asistentes: .....

1. **Asuntos tratados en la reunión.**
   - **Agenda**
     Se hace un resumen narrativo de lo que se hablo en la reunión, se especificaran puntos tratados.
   - **Restricciones (en caso que aplique)**
     Se explicaran las posibles limitaciones, restricciones que se puedan establecer como resultado de lo que se pudo establecer en la reunión.
2. **Pendientes.**
   Se establece que quedara pendiente para la próxima reunión.
3. **Conclusiones**
   Conclusiones a la que se llega con lo que se trato en la reunión.
4. **Próxima reunión.**
   Si en esta reunión se acordó mantener una próxima reunión se podrá la fecha en que se la pacto.
5. **Acuerdos alcanzados**
   Acuerdos que llegaron entre ambas partes.

Los acuerdos alcanzados en esta reunión son la base para el desarrollo del proyecto.

Atentamente,

SDAT	AREA USUARIA
_____	_____
Nombre del técnico Desarrollo de aplicaciones	Nombre del usuario contraparte Área a la que pertenece.

Página 1 de 1

Figura 3-41 Formulario de bitácora de reuniones con el usuario

Firmas de la reunión: finalmente en esta área del formulario se registran las firmas del ingeniero de software responsable y la del usuario contraparte.

El formato para la impresión del formulario de bitácora de reuniones con el usuario se encuentra en el "ANEXO-4.2-Formulario de bitácora de reuniones con el usuario"

3.4.1.3. Documento de estudio de factibilidad

Este documento se debe utilizar en la fase de análisis de requerimientos.

En la figuras 3-42 y 3-43 se visualiza el documento que deben utilizar los ingenieros de software de la SBS, para documentar la factibilidad de desarrollo de la nueva aplicación Web.

Como parte del título de este documento se debe describir el proyecto de automatización, es decir el nombre de la aplicación Web.

Este documento se compone de cuatro secciones:

1. **Identificación del área solicitante:** esta sección contiene la fecha de termino del estudio de la factibilidad, el nombre del área usuaria de la aplicación, el nombre del titular de la área usuaria, el nombre del usuario contraparte y el tiempo en horas de participación en el estudio del usuario contraparte.

2. **Especificaciones del proyecto:** en esta sección se describe el nombre de la aplicación Web, las siglas de identificación de la aplicación Web, la versión de la aplicación Web, los objetivos del proyecto de desarrollo, los principales problemas a resolver, las restricciones y limitantes, los beneficios que se obtendrán, los usuarios principales de la

aplicación, infraestructura de hardware y software necesaria para el funcionamiento de la aplicación, la descripción de las tecnologías a utilizar en el desarrollo, el tiempo aproximado de desarrollo con el respectivo cronograma, el presupuesto referencial.

3. **Descripción conceptual de la aplicación:** resumen narrativo y/o grafico del funcionamiento de la aplicación, con el alcance, limitaciones y riesgos en el proceso de desarrollo de la aplicación Web.

4. **Conclusión sobre la factibilidad del proyecto:** en esta sección se debe describir el criterio técnico del ingeniero de software asignado para el desarrollo de la aplicación Web. El criterio técnico debe estar sujeto a las condiciones de hardware y de software actuales. En el caso de necesitarse más recursos de hardware o de software para el desarrollo el criterio de la factibilidad técnica estará condicionado a la

aceptación de la autoridad correspondiente para la asignación de los nuevos recursos.

En la parte final del documento de estudio de factibilidad se deben registrar las firmas de compromiso del titular del área solicitante, del subdirector de desarrollo y aplicaciones tecnológicas, de los usuarios contrapartes y la firma del ingeniero de software asignado para el desarrollo de la aplicación Web.

ESTUDIO DE FACTIBILIDAD

Proyecto de automatización: (Nombre que el usuario da al requerimiento)

1. Identificación del área solicitante

Fecha del estudio (dd/mm/yyyy): (Fecha en que se finalizo en estudio)

Área usuaria de la aplicación: .....

Nombre del titular del área usuaria: .....

Usuario contraparte: .....

No. de horas comprometidas por el usuario contraparte: (el usuario contraparte pueden ser varios, detallar)

2. Especificaciones del Proyecto

2.1 Nombre del Sistema: .....
2.2 Siglas de identificación: .....
2.3 Versión: .....
2.4 Objetivos del proyecto
- .....
- .....
- .....

2.5 Principales problemas a resolver.
- .....
- .....

2.6 Restricciones, limitantes
- .....
- .....

2.7 Beneficios que se obtendrán
- .....
- .....

2.8 Usuarios principales y roles que cumplen actualmente
- .....
- .....

2.9 Infraestructura de hardware y software necesaria para la solución
- .....
- .....

2.10 Tecnologías de desarrollo

2.11 Tiempo aproximado de desarrollo

(Se adjunta plan de trabajo detallado)

2.12 Presupuesto Referencial

Figura 3-42 Documento de estudio de factibilidad (parte 1 de 2)

ESTUDIO DE FACTIBILIDAD *(Continuación...)*

Proyecto de automatización: (Nombre que el usuario da al requerimiento)

2.12 Presupuesto Referencial

Se calcula en función del costo por remuneraciones del personal involucrado y el tiempo estimado de desarrollo.

3. Descripción conceptual de la aplicación

Se hace un resumen narrativo y/o gráfico de la forma en que funcionaría la aplicación, del alcance y las limitaciones que deben considerarse y de los riesgos que corre la aplicación en el caso de producirse eventos no previstos como cambios de estructura, incumplimiento de los procedimientos acordados, retiro del personal contraparte, etc.

4. Conclusión sobre la factibilidad del proyecto

Se manifiesta un criterio técnico sobre la factibilidad de emprender el proyecto en las condiciones de funcionamiento y recursos actuales.
De requerirse una cantidad de recursos de personal, hardware o software adicionales, para el desarrollo o el futuro funcionamiento de la aplicación, el criterio sobre la factibilidad técnica estaría condicionado a la aceptación de la autoridad para facilitar esos recursos.

Manifestamos nuestra conformidad con lo arriba expuesto.

Atentamente,

Nombre del titular del Área Solicitante Cargo	Usuario Contraparte Área
	Usuario Contraparte Área
Nombre Subdirector de Desarrollo y Aplicaciones tecnológicas Cargo	Informático asignado Desarrollo de Aplicaciones

Figura 3-43 Documento de estudio de factibilidad (parte 2 de 2)

El formato para la impresión del documento de estudio de factibilidad se encuentra en el "ANEXO-4.3-Documento de estudio de factibilidad".

Nota: Los formatos finales, para la impresión de todos los documentos y/o formularios se encuentran en la parte de anexos de esta tesis.

3.4.2. Definición de nomenclaturas y abreviaturas

La existencia de un modelo de nomenclatura coherente, es uno de los elementos más importantes en cuanto a previsibilidad y capacidad de descubrimiento en los elementos de la metodología de desarrollo propuesta. El uso y el conocimiento generalizados de estas instrucciones de nomenclatura y abreviaturas deberían eliminar la mayoría de las preguntas más frecuentes de los ingenieros de software: ¿Dónde está el directorio de imágenes? ¿Cómo puede identificar el contenido de un archivo?, etc.

El detalle de los estándares recomendados para nomenclaturas y abreviaturas se encuentra en el "ANEXO-7-Estándares de nomenclaturas y abreviaturas"

3.5. Matriz de verificación de uso de elementos estándares por fase

Mediante esta matriz se identifican cuáles son los elementos estándares (documentos, diagramas, estándares, anexos) que son obligatorios de utilizar, y cuáles son los opcionales para la correspondiente fase de la metodología de desarrollo.

La matriz de elementos se ha divido de la siguiente manera:

1. Matriz de formularios, documentos, estándares y guías.
2. Matriz de diagramas UML.

3.5.1. Formularios, documentos, estándares y guías recomendados por cada fase de la metodología de desarrollo

ELEMENTOS ESTÁNDARES	FASES DE LA METODOLOGÍA					
	Levantamiento/ investigación de requisitos	Análisis de requisitos	Modelado Conceptual	Elaboración / codificación de componentes	Pruebas	Implementación
Formulario de solicitud de requerimientos	obligatorio	obligatorio				
Formulario de bitácora de reuniones	obligatorio	obligatorio	opcional		opcional	
Documento de Requisitos del Software (DRS)	obligatorio	obligatorio	obligatorio	obligatorio	obligatorio	
Documento de estudio de factibilidad		obligatorio				
Estándares de nomenclaturas y abreviaturas	obligatorio	obligatorio	obligatorio	obligatorio	obligatorio	obligatorio
Estándares de codificación				obligatorio		
Estándares visuales de páginas Web		opcional		obligatorio		
Estándares de instalación y despliegue				obligatorio		obligatorio
Guía para elaborar manual de usuario de la aplicación				obligatorio		
Guía para elaborar manual técnico de la aplicación				obligatorio		

Tabla VIII. Matriz de uso de formularios, documentos, estándares y guías por fase de la metodología

El objetivo de la matriz de uso de los formularios, documentos, estándares y guías en cada fase de la metodología, es brindar un mapa de los recursos que se recomienda utilizar en cada fase de la metodología, y que a su vez sirven como un mecanismo de apoyo para cumplir las metas de cada fase.

La estructura de esta matriz se compone de dos partes, la primera contiene la lista de los formularios, documentos, estándares y guías y la segunda el detalle de cada fase de la metodología de desarrollo. Cada formulario, documento, estándar o guía tiene una relación de uso en cada fase de la metodología de desarrollo.

Un formulario, documento, estándar o guía puede ser utilizado en una o en varias fases. La relación de uso de las dos partes de esta matriz es de dos tipos: obligatorio y opcional.

La relación de uso obligatorio indica que un formulario, documento, estándar o guía es imprescindible en la respectiva fase de la metodología de desarrollo. La relación de uso opcional indica que un formulario, documento, estándar o guía puede o no ser utilizado(a) en la fase de la metodología de desarrollo.

Un formulario, documento, estándar o guía puede tener una relación opcional en las siguientes situaciones:

1. Cuando el proyecto de desarrollo es pequeño, medido desde el número de componentes a desarrollar.

2. Cuando el proyecto de desarrollo tiene un grado de complejidad bajo, medido desde la funcionalidad total.

3. Cuando el ambiente de desarrollo de un proyecto de software no requiere el soporte de muchos documentos. Este caso puede darse cuando los cronogramas del proyecto de desarrollo son muy ajustados en tiempos debido a una solicitud directa de alguna autoridad de turno.

3.5.2. Diagramas UML recomendados por cada fase de la metodología de desarrollo

ELEMENTOS ESTÁNDARES	FASES DE LA METODOLOGÍA					
	Levantamiento/ investigación de requisitos	Análisis de requisitos	Modelado Conceptual	Elaboración / codificación de componentes	Pruebas	Implementación
Diagrama de Casos de Uso	opcional	obligatorio	obligatorio	obligatorio	obligatorio	
Diagrama de Clases y Paquetes		opcional	obligatorio	obligatorio		
Diagrama de Estados			obligatorio	obligatorio	obligatorio	
Diagrama de actividades			obligatorio	obligatorio	obligatorio	
Diagrama de componentes			obligatorio	obligatorio		obligatorio
Diagrama de Despliegue			obligatorio	obligatorio		obligatorio

Tabla IX. Matriz de uso de diagramas UML por fase de la metodología

El objetivo de la matriz de uso de los diagramas UML en cada fase de la metodología, es brindar un mapa de los diagramas que se recomienda utilizar en cada fase de la metodología, y que a su vez sirven como un mecanismo de apoyo para cumplir las metas de cada fase.

La estructura de esta matriz se compone de dos partes, la primera contiene la lista de los diagramas UML y la segunda el detalle de cada fase de la metodología de desarrollo. Cada diagrama UML tiene una relación de uso en cada fase de la metodología de desarrollo.

Un diagrama UML puede ser utilizado en una o en varias fases. La relación de uso de las dos partes de esta matriz es de dos tipos: obligatorio y opcional. La relación de uso obligatorio indica que un diagrama UML es imprescindible en la respectiva fase de la metodología de desarrollo.

La relación de uso opcional indica que un diagrama UML puede o no ser utilizado en la fase de la metodología de desarrollo. Un diagrama UML puede tener una relación opcional en las siguientes situaciones:

1. Cuando el proyecto de desarrollo es pequeño, medido desde el número de componentes a desarrollar.

2. Cuando el proyecto de desarrollo tiene un grado de complejidad bajo, medido desde la funcionalidad total.

3. Cuando el ambiente de desarrollo de un proyecto de software no requiere el soporte de muchos documentos. Este caso puede darse cuando los cronogramas del proyecto de desarrollo son muy ajustados en tiempos debido a una solicitud directa del alguna autoridad de turno.

3.6. Planificación de las fases

Cuando se planifica un proyecto se tiene que obtener estimaciones del costo y esfuerzo humano requerido por medio de las mediciones de

software que se utilizan para recolectar los datos cualitativos acerca del software y sus procesos para aumentar su calidad. En la mayoría de los desafíos técnicos, las métricas nos ayudan a entender tanto el proceso técnico que se utiliza para desarrollar un producto, como el propio producto. El proceso para intentar mejorarlo, *el producto se mide para intentar aumentar su calidad.*

El principio, podría parecer que la necesidad de la medición es algo evidente. Después de todo es lo que nos permite cuantificar y por consiguiente gestionar de forma más efectiva. Pero la realidad puede ser muy diferente. Frecuentemente la medición conlleva una gran controversia y discusión.

1. ¿Cuáles son las métricas apropiadas para el proceso y para el producto?
2. ¿Cómo se deben utilizar los datos que se recopilan?
3. ¿Es bueno usar medidas para comparar gente, procesos o productos?

Estas preguntas y otras tantas docenas de ellas siempre surgen cuando se intenta medir algo que no se ha medido en el pasado.

3.6.1. Recomendaciones en la planificación de proyectos

3.6.1.1. Los horarios del proyecto

Un rendimiento importante de proyecto es planear el horario del proyecto, junto con horarios de apoyo que son una representación de tiempo gráfica de todas las actividades relacionadas necesarias. El horario del proyecto establece los parámetros de tiempo del proyecto y auxilios a los gerentes para coordinar eficazmente y facilitar los esfuerzos del equipo del proyecto entero durante la vida del proyecto. Un horario se vuelve una parte eficaz del sistema de mando de proyecto. Ser eficaz, debe ser un buen horario del proyecto.

- Entendible por el equipo del proyecto.

- Capaz de identificar y resaltar paquetes de trabajo críticos y tareas.

- Puesto al día, modificaciones como necesarias, y flexible en su aplicación.

- Substancialmente detallado para mantener una base comprometiendo, supervisando, y evaluando el uso de recursos del proyecto.

- Basado en el tiempo creíble que estima, eso conforme a los recursos disponibles.

- Compatible con otros planes orgánicos que comparten los recursos comunes.

3.6.1.2. Fijando las Técnicas

Varias técnicas de planificación son útiles tratando con el aspecto cronometrando del recurso del proyecto.

3.6.1.2.1. Los proyectos de planificación con barra de mapas

Una técnica para la planificación del proyecto simple es fijar lo basado en el mapa de la barra. Este mapa consiste en una balanza dividida en las unidades del tiempo (por ejemplo: día, semana, mes) y una inscripción de los paquetes de trabajo del proyecto o elementos. Se usan barras o líneas para indicar el horario y estado de cada paquete de trabajo respecto a la balanza de tiempo.

3.6.1.2.2. Diagramas de GANTT

Este tipo de diagrama es particularmente fácil de implementar con una simple hoja de cálculo, pero también existen herramientas especializadas, la más conocida es **Microsoft Project**. Debido al contrato de licencias de Microsoft Project en la SBS se recomienda utilizar este software para planificar los proyectos de desarrollo de aplicaciones Web. En un futuro se puede evaluar el reemplazo de software de ofimática con

licencias por el equivalente software libre. Un ejemplo de un diagrama de GANTT se puede ver en la figura 3-44.

Figura 3-44 Diagrama de GANTT del Sistema de Providencias Judiciales

3.6.2. Asignación del recurso humano

Todo proyecto depende de recursos humanos. Si el *project manager* no sabe administrar personal (ya sea por inexperiencia o incapacidad) recae en el arquitecto o líder técnico la responsabilidad de administrar dichos recursos de forma eficiente para llevar a buen término el proyecto. Sin entrar en discusiones acerca de teorías de psicología industrial, el mejor consejo a seguir es: "*el mejor administrador es aquél que guía a su gente y la deja trabajar*". Un buen libro de referencia es Peopleware **[48][49].**

3.6.2.1. Errores comunes en la asignación del recurso humano

Las siguientes recomendaciones van dirigidas a los jefes de proyectos de aplicaciones Web de la SBS. A continuación se presentan los errores más comunes al momento de asignar el recurso humano a un proyecto de software:

1. Mal análisis en los requerimientos.
2. Una mala planeación del recurso humano para el desarrollo de los módulos de la aplicación Web.
3. No hacer un análisis costo beneficio de aplicar tecnologías fuera del dominio del recurso humano.
4. Desconocer las verdaderas capacidades de desarrollo del recurso humano.

3.6.3. Planificación de los costos del proyecto por fases

Se recomiendan en las siguientes secciones, algunas ayudas al momento de realizar la estimación de tiempos y costos en desarrollos de software.

3.6.3.1. Estimación

Cuando se planifica un proyecto de software se tiene que obtener estimaciones de esfuerzo humano requerido, de la duración cronológica del esfuerzo humano requerido, de la duración cronológica del proyecto y del costo.

Se han desarrollado varias técnicas de estimación para el desarrollo de software, aunque cada una tiene sus puntos fuertes y sus puntos débiles, todas tienen en común los siguientes atributos:

1. Se ha de establecer de antemano el ámbito del proyecto.

2. Como bases para la realización de estimaciones se usan métricas del software de proyectos pasados.

3. El proyecto se desglosa en partes más pequeñas que se estiman individualmente.

3.6.3.2. Planeación del proyecto

La planeación efectiva de un proyecto de software depende de la planeación detallada de su avance, anticipando problemas que puedan surgir y preparando con anticipación soluciones tentativas a ellos. El administrador del proyecto será responsable de la planeación desde la definición de requisitos hasta la entrega del sistema terminado.

Los puntos analizados posteriormente generalmente son requeridos por grandes sistemas de programación, sin embargo estos puntos son válidos también para sistemas pequeños.

- Plan de fases.
- Plan de organización.
- Plan de pruebas.
- Plan de control de modificaciones.
- Plan de documentación.

- Plan de capacitación.
- Plan de revisión e informes.
- Plan de instalación y operación.
- Plan de recursos y entregas.
- Plan de mantenimiento.

3.6.4. Planificación de tiempos de las actividades por cada fase

En base a la aplicación de métricas se recomienda realizar una estimación de tiempos para las actividades. Es importante en este sentido hacer referencia a las métricas de software: a las técnicas, de calidad, de productividad, orientadas a la persona, y de las métricas orientadas al tamaño, porque mediante ellas se podría predecir ciertas variaciones del tiempo asignado a cada actividad.

Las métricas de software que se recomiendan utilizar en el desarrollo de aplicaciones Web en la SBS son las siguientes:

- Métricas técnicas.

- Métricas de calidad.
- Métricas de productividad.
- Métricas orientadas a la persona.
- Métricas orientadas al tamaño: Se pueden aplicar las siguientes fórmulas:

  Productividad = KLDC (miles de líneas de código)/persona-mes (persona-mes es el esfuerzo)

  Calidad = errores/KLDC

  Documentación = Págs. Doc./KLDC

  Costo = $/KLDC

- Métricas orientadas a la función: En lugar de calcularlas las LDC, las métricas orientadas a la función se centran en la funcionalidad o utilidad del programa.

  Los puntos de función se calculan rellenando la tabla como se muestra en la figura 3-45. Para calcular los puntos de función se utiliza la siguiente relación: PF = CUENTA_TOTAL * [0.65 + 0.01 * SUM(fi)]

Parámetro de medición	Cuenta		Simple	Medio	Complejo	
			FACTOR DE PONDERACIÓN			
Número de entradas de usuario		x	3	4	6	=
Número de salidas de usuario		x	4	5	7	=
Número de peticiones de usuario		x	3	4	6	=
Número de archivos		x	7	10	15	=
Número de interfaces externas		x	5	7	10	=
Cuenta = Total						

Figura 3-45 Factores de ponderación para el punto de función

Donde CUENTA_TOTAL es la suma de todas las entradas de PF obtenidas de la figura 3-45. Fi donde i puede ser de uno hasta 14 los valores de ajuste de la complejidad basados en las respuestas a las cuestiones señaladas en la figura 3-46.

0	1	2	3	4	5
Sin influencia	Incidental	Moderado	Medio	Significativo	Esencial

fi:

1. ¿Requiere el sistema copias de seguridad y recuperación fiables?
2. ¿Se requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es crítico el rendimiento?
5. ¿Será ejecutado el sistema en un entorno operativo existente y frecuentemente utilizado?
6. ¿Requiere el sistema entrada de datos interactivo?
7. ¿Requiere la entrada de datos interactivo que las transiciones de entrada se lleven a cabo sobre múltiples o variadas operaciones?
8. ¿Se actualizan los archivos maestros en forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidos en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado el sistema para facilitar los cambios y para ser fácilmente utilizado por el usuario?

gura 3-46 Preguntas para valores de ajustes del punto de función

Una vez calculado los puntos de función se usan de forma analógica a las LDC como medida de la productividad, calidad y otros productos del software.

- Productividad = PF/persona-mes
- Calidad = errores/PF
- Costo = dólares/PF
- Documentación = Págs.Doc./PF

.6.5. Seguimiento y control del cumplimiento de actividades por cada fase del proyecto

Se recomienda realizar actividades de seguimiento y control del cumplimiento de las actividades por cada fase de los proyectos de software, ya que de esta forma podemos ayudar a asegurar que el equipo de personas participantes, cumple satisfactoriamente con las metas del plan del proyecto.

Las actividades de seguimiento y control deben ser realizadas por el líder del proyecto y por los líderes de equipos. Las principales variables de un proyecto de desarrollo de software son:

- Costo
- Tiempo
- Rendimientos (Performance)

3.6.5.1. Tareas de seguimiento y control

Las tareas recomendadas para ejecutar un proceso ordenado de seguimiento y control son las siguientes:

1. Seguir y revisar los resultados y logros del proyecto.
2. Revisar el Plan de Proyecto para:
   - Reflejar los resultados obtenidos.
   - Ajustar las tareas restantes si es necesario.
3. Analizar el progreso en la ejecución del plan.
4. Tomar acciones correctivas en caso de desvíos.

5. Fijar nuevas metas.

3.6.5.2. Documento entregables de seguimiento y control

Los documentos que se deben elaborar en el proceso de seguimiento y control son los siguientes:

1. Reportes de avance o estado actual.
2. Actualizar la lista de tareas, riesgos y problemas.
3. Actualizar el plan y el cronograma, a fin de reflejar los avances.
4. Auditoria y reporte de los ítems en desarrollo.

3.7. Evaluación de resultados por cada fase

La evaluación de los resultados de cada fase, deberá realizarse mediante la matriz de objetivos alcanzados, propuesta en la siguiente sección. Luego de llenar esta matriz se deberán realizar las tareas de retroalimentación con el usuario.

3.7.1. Matriz de objetivos alcanzados

La matriz de objetivos alcanzados tiene como finalidad mostrar un mapa que describe la relación de los objetivos de cada fase de la metodología de desarrollo con los problemas que se presenten en el cumplimiento de cada objetivo.

En base a los problemas encontrados para el cumplimiento de cada uno de los objetivos por fase se pueden realizar tareas de retroalimentación. Mediante las tareas de retroalimentación se pueden establecer mecanismos de solución a los diferentes problemas que se hayan presentado en cada fase. Las técnicas de retroalimentación sugeridas se presentan más adelante. En la tabla X se puede observar el formato de la matriz de objetivos alcanzados, clasificados por fases, actividades operativas. En esta matriz se deben asignar valores a la columna de indicador del cumplimiento de la actividad, y a la columna de los problemas encontrados para el desarrollo de la respectiva actividad.

OBJETIVO ESPECÍFICO	ACTIVIDADES OPERATIVAS	META ALCANZADA		PROBLEMAS ENCONTRADOS EN EL CUMPLIMIENTO DE LAS ACTIVIDADES
		SI	NO	
FASE: *ANÁLISIS DE REQUERIMIENTOS*				
Capturar los requisitos del usuario	Realizar entrevistas, sesiones JAD o sesiones de brainstorm			
Estructurar los requisitos	Elaborar el Documentos de Requisitos del Software			
Identificar a los usuarios responsables	Elaborar diagramas de casos de uso UML			
	Elaborar diagrama de clases preliminar UML			
Preparar diseño visual (prototipos)	Elaborar prototipos de acuerdo a estándares visuales Web			
Documentar las necesidades funcionales	Elaborar plantillas de casos de uso			
FASE: *MODELADO CONCEPTUAL*				
Definir los límites de la aplicación	Realizar el diagrama de componentes UML			
Definir los objetos del mundo real con sus atributos y relaciones	Elaborar diagramas de clases, paquetes, secuencia y estados UML			
Clasificar los componentes de la aplicación	Elaborar diagrama de componentes UML			
Especificar el comportamiento de la aplicación	Realizar diagramas de secuencia UML			
FASE: *ELABORACIÓN DE COMPONENTES*				
Crear componentes reusables y reemplazables	Realizar la tarea de codificación basada en el documento de estándares de lenguaje Java			
Implementar las interfaces de los componentes	Realizar la tarea de codificación basada en el documento de estándares de lenguaje Java			
Refinar el diseño visual de las aplicaciones Web	Elaborar ajustes de las pantallas del prototipo de la aplicación Web, según los estándares visuales definidos			
Documentar los componentes	Generar el documento de especificación de componentes en el formato JAVADOC			
Probar el funcionamiento de la aplicación	Realizar pruebas de la aplicación mediante los casos de uso			
Especificar las instrucciones de instalación de la aplicación	Elaborar el manual de instalación o despliegue de la aplicación Web			
Dar instrucciones para el manejo de la aplicación Web al usuario	Redactar el manual de usuario de la aplicación Web			
Generar instrucciones del funcionamiento técnico de la aplicación Web	Elaborar el manual técnico de la aplicación Web			
FASE: *PRUEBAS E IMPLEMENTACIÓN*				
Verificar que la aplicación Web cumpla con los requisitos del análisis y diseño	Elaborar casos de prueba basados en los casos de uso y realizar las verificaciones			
Ajustar los errores y defectos de la aplicación Web	Realizar modificaciones en el código fuente			
Asegura la calidad de la aplicación Web	Ejecutar las pruebas a todos los módulos de la aplicación Web			
Revisar cambios en los manuales de instalación, técnico y de usuario	Realizar ajustes a los manuales de instalación, técnico y de usuario, si es que la aplicación Web sufre algún cambio			

Tabla X. Matriz de objetivos alcanzados

3.7.2. Retroalimentación del plan de cada fase

Éste es un punto importante para resaltar, aumenta mucho la socialización y comunicación entre los distintos equipos relacionados en el proyecto, es recomendado manejar una retroalimentación constante y oportuna en todas las fases del proyecto de forma rápida y clara.

El cliente es el usuario final encargado de administrar, manejar y obtener el máximo provecho del sistema desarrollado, por lo tanto es recomendable que los aportes de este usuario tengan eco y sean primordiales en las fases: Pruebas, Entregas frecuentes, versionado e integración para obtener como resultado un óptimo producto final y aprobado por el cliente.

Algunos factores que se deben tomar en consideración para reforzar y asumir los posibles cambios generados luego de cada fase son los siguientes:

- Valentía.
- Comunicación.
- Sencillez.

3.7.2.1. Técnicas de retroalimentación

Entre algunas de las técnicas utilizadas para realizar retroalimentación de los problemas encontrados en cualquier fase de la metodología de desarrollo se mencionarán las más importantes.

Se debe hacer un afinamiento de los requerimientos, análisis y diseño, implementación, pruebas, evaluación de resultados y retroalimentación:

- Taller técnico (borradores, retroalimentación)
- Generamos la documentación y la ponemos a disposición de los interesados.
- Iniciamos el desarrollo de prototipos.

- Pruebas, evaluación y retroalimentación.
- Actualización de la documentación.

3.8. Ventajas y desventajas

3.8.1. Ventajas de la metodología

- Mitigación temprana de posibles riesgos altos.
- Progreso visible en las primeras etapas.
- Temprana retroalimentación que se ajuste a las necesidades reales.
- Gestión de la complejidad.
- El conocimiento adquirido en una iteración puede aplicarse de iteración a iteración.
- Abordar las cuestiones de alto riesgo y valor en las primeras iteraciones.
- Usuarios involucrados continuamente.
- Verificar continuamente la calidad desde el principio y con frecuencia.
- Aplicar casos de uso.
- Modelar el Software visualmente.

- Gestión cuidadosa de requisitos.
- Control de cambios.
- Fácil de entender.
- Fácil de aplicar.
- Sirve de base para dar soporte a obtener certificaciones con modelos o normas, tales como normas ISO o el modelo propuesto CMM.
- Permite un desarrollo iterativo y adaptable que permite la integración de nuevas funcionalidades a lo largo del desarrollo del proyecto.
- El producto final tiene una calidad adecuada.
- Es incremental: entregas pequeñas de software, con ciclos rápidos.
- Es cooperativo: cliente y desarrolladores trabajan juntos constantemente con una cercana comunicación.
- Es sencilla: el método en sí mismo es simple, fácil de aprender y modificar.
- Está bien documentada y es adaptable. Permite organizar los cambios de último momento.

- Producir versiones ejecutables del software en pocas semanas, con el fin de quedar bien con el cliente y obtener retroalimentación en cuanto al producto.
- Inventar soluciones sencillas, de tal forma que el impacto de los cambios se minimice.
- Mejorar la calidad del diseño de manera continua, haciendo que la siguiente iteración requiera menos esfuerzo que la actual.
- Probar desde temprano y de manera continua, para encontrar defectos antes de que tengan un alto impacto.
- Software funcional por encima de documentación abundante.
- Colaboración con los clientes por encima de negociación de contratos.
- Respuesta al cambio por encima de seguimiento a planes.
- Seguridad en el desenlace del proyecto.
- Eficiencia en el desarrollo.
- Habitabilidad de las reglas (el equipo se siente cómodo con ellas).
- Reducción de riesgos basado en la retroalimentación temprana.
- Pruebas continuas e iterativas promueven una mejor evaluación del estado del proyecto.
- Los patrocinadores reciben evidencia concreta del avance del proyecto.

- Se pueden acomodar mejor los cambios (requerimientos, tácticos y tecnológicos).
- Los problemas más complejos se atacan primero.
- Permite "administrar la complejidad" del proyecto, debido a la resolución de las partes.
- Demuestra un progreso consistente y temprano en el desarrollo.
- Abarca las tareas de riesgos.
- Apoya el desarrollo basado en componentes (nuevos o existentes).
- Permite rastrear los cambios.
- Se puede utilizar en proyectos grandes y pequeños.
- Asegura la calidad.

3.8.2. Desventajas de la metodología

- Requiere conocimientos del proceso y de UML.
- Se corre el riesgo de que los usuarios, acostumbrados al sistema anterior, demoren excesivamente la prueba del nuevo, ya que no cuentan con la exigencia de un plazo a término inamovible. El sistema actual lo conocen y dominan bien, el nuevo desarrollado

con las directrices de la nueva metodología es un cambio y un desafío.

- Solo se "pierde" el trabajo de una iteración.
- Se hace difícil de gestionar sino se cuenta con el apoyo de los directivos de las áreas de la SBS.
- Se pueden descubrir nuevos riesgos durante el camino, como por ejemplo alguna innovación tecnológica del software base.
- Si en el uso de la metodología no se aplica los elementos estándares obligatorios por fase, entonces no se garantiza el éxito de su uso.
- Hay costos ocultos en su implementación, ya que se incorporan varias actividades a realizar por el equipo, y hay que saber medir ese impacto para no fracasar en el intento.

CAPÍTULO IV

"CAPACITACIÓN"

CAPÍTULO IV

4. CAPACITACIÓN

4.1. Definición del plan de capacitación de la metodología de desarrollo

Este plan busca dotar a los ingenieros de software de la SBS, de los conocimientos y las destrezas necesarias para convertirse en componentes productivos de un equipo de desarrollo sobre la metodología de desarrollo propuesta. La capacitación en el uso de esta metodología, es un proceso educacional de carácter estratégico, que aplicado de manera organizada y sistémica, el personal técnico adquirirá o desarrollará conocimientos y habilidades específicas relativas a la organización de su trabajo, y a su vez modificar sus actitudes frente a aspectos del ambiente laboral.

Esta capacitación implica por un lado, una sucesión definida de condiciones y etapas orientadas a lograr la integración del colaborador a su puesto y a la organización, el incremento y mantenimiento de su eficiencia, así como su progreso personal y laboral en la SBS. El Plan de Capacitación incluye a los ingenieros de software de la SDAT oficina Quito y las oficinas regionales (Guayaquil, Cuenca y Portoviejo) que integran a la SBS, agrupados de acuerdo a turnos para no descontinuar sus actividades laborales. Algunos de los turnos fueron recogidos de la sugerencia de los propios colaboradores.

4.1.1. Fines del plan de capacitación

Siendo su propósito general impulsar la eficacia organizacional, la capacitación se lleva a cabo para contribuir a:

- Elevar el nivel de rendimiento de los colaboradores de la SDAT y, con ello, al incremento de la productividad y rendimiento de la Dirección Nacional de Recursos Tecnológicos en la SBS.

- Mejorar la interacción entre los colaboradores de la SDAT y, con ello, a elevar el interés por el aseguramiento de la calidad en los servicios que prestan las aplicaciones Web.

- Satisfacer más fácilmente requerimientos futuros de las unidades de la SBS en materia de aplicaciones Web, sobre la base de la planeación de recursos humanos, de software y de hardware.

- Generar conductas positivas y mejoras en el clima de trabajo, la productividad y la calidad y, con ello, a elevar la moral de trabajo.

- La compensación indirecta, especialmente entre las unidades administrativas de la SBS, que tienden a considerar así la paga que asume la SBS para su participación en este programa de capacitación.

- Mantener al colaborador al día con los avances tecnológicos, lo que alienta la iniciativa y la creatividad y ayuda a prevenir la obsolescencia de la fuerza de trabajo.

4.1.2. Objetivos del plan de capacitación

Los **objetivos generales** del plan de capacitación son los siguientes:

- Preparar al personal para la ejecución eficiente de sus responsabilidades que asuman en los puestos de la SDAT.

- Brindar oportunidades de desarrollo personal en los cargos actuales y para otros puestos de la SDAT, en los que el colaborador puede ser considerado.

- Modificar actitudes para contribuir a crear un clima de trabajo satisfactorio, incrementar la motivación del trabajador y hacerlo más receptivo a la supervisión y acciones de gestión.

Los **objetivos específicos** a seguir en el plan de capacitación son:

- Proporcionar orientación e información relativa a los objetivos de la metodología de desarrollo, su organización, funcionamiento, procedimientos y políticas.

- Proveer conocimientos y desarrollar habilidades que cubran la totalidad de requerimientos para el desempleo de las funciones de puestos específicos de la SDAT.

- Actualizar y ampliar los conocimientos requeridos en las capacitaciones recibidas respecto de la actividad de desarrollo de aplicaciones Web.

- Contribuir a elevar y mantener un buen nivel de eficiencia individual y rendimiento colectivo mediante la organización del trabajo.

- Ayudar en la preparación de personal calificado, acorde con los planes, objetivos y requerimientos de las unidades de la SBS.

- Apoyar la continuidad y desarrollo institucional.

- Entender las técnicas específicas de la metodología, los procedimientos organizativos y de gestión que la hacen efectiva.

4.1.3. Estrategias para la capacitación

Las estrategias a emplear son:

- Desarrollo de trabajos prácticos que se vienen realizando cotidianamente.
- Presentación de caso de estudio.
- Realizar talleres.
- Metodología de exposición – diálogo.

4.1.4. Tipos de capacitación

Los tipos de capacitación que se realizarán en la SBS son:

- Capacitación Inductiva.
- Capacitación Preventiva.
- Capacitación Correctiva.
- Capacitación para el Desarrollo de Carrera.

4.1.5. Acciones a desarrollar

Las acciones para el desarrollo del plan de capacitación están respaldadas por los temarios que permitirán a los asistentes a capitalizar los temas, y el esfuerzo realizado que permitirán mejorar

la calidad de los recursos humanos de la SDAT, para ello se está considerando lo siguiente:

TEMAS DE CAPACITACIÓN

Conceptos de la metodología de desarrollo

- ¿Cuáles son las fases de la metodología?
- ¿Qué se debe hacer en cada fase de la metodología?
- Documentos y estándares aplicados por fase

Caso de estudio

- Consideraciones en el uso de la metodología
- Automatización de la notificación de providencias judiciales

4.1.6. Cronograma de la capacitación

Figura 4-1 Cronograma de la capacitación de la metodología

En la figura 4-1 se puede observar el cronograma de capacitación de la metodología de desarrollo. De acuerdo a los temas de capacitación se establece un cronograma con tiempo de duración en días estimados. Para asignar los tiempos de duración de en días de cada tarea del cronograma, se tomó como punto de partida al personal de nivel intermedio, ya que cuentan con la capacitación de modelamiento en UML y de programación en Java.

Las modalidades de capacitación serán las de actualización, perfeccionamiento y complementación. El tipo de capacitación será realizada de la forma inductiva.

4.2. Estudio de casos

Lo que se pretende con la aplicación de esta metodología, es desarrollar y mantener aplicaciones Web, participando en el diseño y documentación de las mismas, de conformidad con los requisitos funcionales, a través de las tecnologías ya expuestas, con el fin de dar

respuesta a la alta demanda de las unidades administrativas de la SBS, dando cobertura a los requerimientos del sector informático para el desarrollo de aplicaciones Web.

En este mismo camino, la SDAT dio todo el apoyo para adoptar esta nueva metodología de desarrollo. La SDAT dispuso un proyecto pequeño de desarrollo perteneciente a la unidad Secretaria General de la SBS. Este proyecto de desarrollo consiste en automatizar el proceso de notificaciones de providencias judiciales a las entidades controladas del sistema financiero.

En el capítulo 6 se presenta el detalle de la aplicación de la metodología de desarrollo a este proyecto de software. El caso de estudio está estructurado de la siguiente forma:

1. Recepción de la solicitud de requerimiento de desarrollo de la aplicación Web.
2. Ejecución de la fase de análisis incluidos todos los recursos.

3. Elaboración de modelos o mapas de la aplicación Web en la fase de modelado conceptual.
4. Guías de cómo se realizó la codificación de los componentes Java, la elaboración del manual de instalación, técnico y de usuario en la fase de elaboración de componentes.
5. Finalmente como se realizó la fase de pruebas e implementación de la aplicación Web.

4.2.1. Objetivos del caso de estudio

Los objetivos que se esperan obtener en el caso estudio de la aplicación de la metodología de software, son los siguientes:

- Enseñar al personal de la SDAT nuevas técnicas organizadas de programación orientada a objetos, utilizando algunas de las tecnologías más competentes del mercado, como es la programación en el entorno de Oracle ADF.

- Que el personal de la SDAT obtenga nuevos conocimientos de programación Web, organizando el desarrollo de páginas dinámicas con acceso a bases de datos bajo la arquitectura del Oracle Application Server, utilizando los lenguajes de programación Web más comunes en la actualidad como son Java, JavaScript y una introducción al XML.

- Otorgar al personal de la SDAT las guías para elaborar un diseño organizado de interfaces entre el usuario y el ordenador utilizando el entorno de desarrollo de Oracle ADF y además reducir la brecha del conocimiento del personal de la SDAT, con respecto al trabajo organizado en el proceso de desarrollo de aplicaciones Web.

4.3. Consideraciones en el uso de la metodología de desarrollo

Al momento de aplicar las guías proporcionadas en la metodología de desarrollo se deben tener en cuenta las siguientes acciones:

- Cada aproximación al desarrollo de software realizada mediante esta metodología está basada en objetivos por fase, no debe perderse por ninguna circunstancia el camino a seguir.

- Es importante que al aplicar las herramientas de la metodología, se desarrolle el software de manera coherente, para que luego se fusionen los productos de cada fase y de esta manera poder moverse no sólo hacia adelante en el ciclo de vida del desarrollo, sino hacia atrás de forma que se pueda comprobar el trabajo realizado y se puedan efectuar correcciones.

- En todas las fases de la metodología de desarrollo debe haber una comunicación efectiva entre analistas, programadores, usuarios y gestores, para realizar progresos visibles durante cada actividad del desarrollo.

CAPÍTULO V

"COMPARACIÓN CON OTRAS METODOLOGÍAS"

CAPÍTULO V

5. COMPARACIÓN CON OTRAS METODOLOGÍAS

En este capítulo se presenta la comparación de la metodología de desarrollo propuesta con otras metodologías de desarrollo orientadas a objetos, mostrando cuáles son las ventajas y desventajas entre utilizar una u otra metodología. Se propone revisar tres tipos de comparaciones entre las metodologías orientadas a objetos, metodologías agiles y la metodología de desarrollo propuesta:

1. Comparación de características y fases.
2. Comparación de componentes utilizados en cada fase.
3. Comparación de productos generados en cada fase.

Para realizar la comparación de la metodología propuesta, se hace referencia en este capítulo a 3 metodologías de desarrollo de software. Dos metodologías orientadas a objetos y una metodología de desarrollo ágil. Las metodologías orientadas objetos seleccionadas para efectuar la comparación de metodologías son:

a) Rational Unified Process (RUP) – Proceso Unificado de Rational
b) Object-oriented software process (OOSP)

Y la metodología de desarrollo de software ágil escogida es la de Programación Extrema (Xtreme Programming – XP).

Las tres metodologías de desarrollo de software fueron escogidas por tener muchas partes parecidas a las de la metodología de desarrollo propuesta. De esta manera se pudo realizar la comparación mencionada, encontrando las similitudes y diferencias en la aplicación de cada una de las metodologías de desarrollo. Mediante las comparaciones realizadas se puede hacer un juicio de valor respecto a las características que debe tener una metodología de

desarrollo que se adapte de forma sencilla y rápida al proceso de desarrollo de software existente en la Superintendencia de Bancos y Seguros.

5.1. Comparación de fases de las metodologías

En esta sección se realiza una comparación de las características y de las fases que se manejan en cada metodología de desarrollo.

En la tabla XI se recoge esquemáticamente las principales diferencias de la metodología de desarrollo propuesta con respecto a las metodologías orientadas a objetos y metodologías agiles, en lo que se refiere a sus características.

Características	Metodología Propuesta	Rational Unified Process	Object-oriented software process (OOSP)	Xtreme Programing
Basadas en normas provenientes de estándares	Total	Total	Parcial	Parcial
El cliente interactúa con el equipo de desarrollo mediante reuniones	Total	Total	Total	Parcial
La arquitectura del software es esencial y se expresa mediante modelos	Total	Total	Parcial	Parcial
Muchos artefactos o componentes. El modelado es esencial, mantenimiento de modelos	Total	Total	Parcial	Parcial
Aplicable a proyectos de cualquier tamaño, pero suele ser especialmente efectiva/usada en proyectos grandes y con equipos posiblemente dispersos	Total	Parcial	Parcial	No aplica
Responde al cambio sobre el seguimiento del plan del proyecto de desarrollo	Parcial	Parcial	Parcial	Total
La gente de negocio y los desarrolladores trabajan juntos diariamente a lo largo del proyecto	Parcial	Parcial	Parcial	Total
Promueve el desarrollo sostenible, es decir, que los patrocinadores, desarrolladores y usuarios son capaces de mantener un ritmo constante indefinidamente	Parcial	Parcial	Parcial	Total
A intervalos regulares, el equipo reflexiona acerca de cómo ser más efectivo, entonces realiza los ajustes y modifica su comportamiento en consecuencia	Total	Parcial	Parcial	Parcial
Maneja excesiva documentación	Parcial	Total	Total	No aplica
Es mejor si hay requerimientos mal definidos	Parcial	Parcial	Parcial	Total
No produce documentación del código y usa revisiones informales	No aplica	No aplica	No aplica	Total
Requiere mucha disciplina y un buen gerente de proyecto de desarrollo	Parcial	Parcial	Parcial	Total
Maneja jerarquía de equipos y permite mejor repetitividad	Total	Parcial	Parcial	No aplica
Incluye seguimiento y reporte del estado del proyecto: mantiene calmados a los dueños del nuevo software	Total	Parcial	Parcial	Parcial

Tabla XI. Matriz de comparación de características de las metodologías de desarrollo

La matriz planteada en la tabla XI, presenta la comparación entre las cuatro metodologías de desarrollo de software, que se basa en una estructura que destaca por un lado las características de cada una de las metodologías y por otro muestra los niveles de aplicación de cada característica en la correspondiente metodología de desarrollo.

Los niveles se dividen en: total, parcial y no aplica. El nivel más alto es el de **total** que indica un alto nivel de aplicabilidad de cierta característica. El nivel intermedio es el **parcial**, el que demuestra que la metodología posee en ciertas ocasiones la respectiva característica, como por ejemplo cuando se trata de proyectos pequeños y simples. Finalmente el nivel **no aplica**, indica que la característica no forma parte de la metodología de desarrollo en ninguna de las fases que la componen.

Fases	Metodología Propuesta	Rational Unified Process	Object-oriented software process (OOSP)	Xtreme Programing
Dominio de Análisis	5	5	4	3
Requerimientos en el análisis	4	3	3	2
Diseño	4	4	3	2
Implementación	3	3	3	1
Pruebas	3	4	4	1

Significado de valores:

- 0 indica que no se hace ninguna mención.
- 1 indica que la mención está hecha, pero no se proporciona ningún detalle.
- 2 indica que la mención está hecha y una definición está provista.
- 3 indica que la mención está hecha, una definición, y por lo menos se presenta un ejemplo.
- 4 indica que la mención está hecha, una definición, y por lo menos se presenta un ejemplo, y se define un proceso.
- 5 indica que la mención está hecha, una definición, por lo menos se presenta un ejemplo, se define un proceso, y se provee la heurística.

Tabla XII. Matriz de comparación de fases aplicadas en las metodologías de desarrollo

El cuadro comparativo indicado en la matriz de la tabla XII, presenta las fases de las metodologías de desarrollo en primer lugar, para luego asignarles un valor dentro de la escala numérica desde el 1 al 5.

Cada valor de la escala numérica representa un nivel de los trabajos que se realizan por cada una de las fases de la respectiva metodología de desarrollo de software.

La interpretación que se aconseja dar a esta matriz de comparación, es la de limitarse a confrontar los trabajos y/o tareas que tienen cada una de las metodologías de desarrollo en general.

Finalmente luego de observar la suma de los valores de las escalas numéricas de cada metodología, se puede concluir que la metodología de desarrollo propuesta conlleva un nivel alto de tareas por realizar, lo cual no indica que sea malo, y por el contrario indica que la metodología de desarrollo propuesta se sujeta a las necesidades actuales de la Superintendencia de Bancos y Seguros

5.2. Comparación de componentes utilizados

Para efectos de realizar esta comparación se consideran como componentes a las herramientas que sirven como soporte a cada una de las fases o a varias fases correspondientes a las metodologías de desarrollo de software. En esta sección se presenta una comparación entre las herramientas utilizadas en la metodología de desarrollo propuesta y las herramientas que se utilizan en otras metodologías de desarrollo. La descripción de las herramientas o componentes que son de uso común entre la metodología de desarrollo propuesta y otras metodologías de desarrollo se presenta en el apartado "5.2.1. Componentes comunes".

5.2.1. Componentes comunes

La siguiente lista muestra los componentes comunes que sirven para realizar la comparación entre las metodologías escogidas y la metodología propuesta:

1. Documento de especificación de requerimientos
2. Modelo conceptual o diseño de la aplicación
3. Componentes de sistema
4. Componentes de negocio
5. Librerías de re-uso portables
6. Diagramas de interacción de componentes
7. Diagrama de arquitectura de la aplicación
8. Documentación del código fuente
9. Documentos de prueba de la aplicación
10. Guías para la instalación de la aplicación

5.2.2. Matriz de comparación de componentes utilizados

En la tabla XIII se presentan las principales similitudes y diferencias de la metodología de desarrollo propuesta con respecto a las metodologías orientadas a objetos y metodologías agiles, en lo que se refiere a los componentes o herramientas que utilizan respectivamente.

Componentes o herramientas	Metodología Propuesta	Rational Unified Process	Object-oriented software process (OOSP)	Xtreme Programing
Documento de especificación de requerimientos	Documento integrado	Diagramas y documento dispersos	Diagramas y documento dispersos	No aplica
Modelo conceptual o diseño de la aplicación	Diagramas UML y Modelo E/R	Diagramas UML	Diagramas	Modelo E/R
Componentes de sistema	Diagramas UML	Diagramas UML	Diagramas	Uso parcial
Componentes de negocio	Diagramas UML y Mapa estático del negocio	Diagramas UML	Diagramas	Uso parcial
Librerías de re-uso portables	Total portabilidad	Depende del lenguaje de programación utilizado	Parcial	Parcial
Diagramas de interacción de componentes	Diagramas UML	Diagramas UML	Diagramas	Parcial
Diagrama de arquitectura de la aplicación	Diagrama UML	Diagrama UML	No aplica	Parcial
Documentación del código fuente	En formato Web	Reglas UML	Documento técnico hecho a medida	Documentación técnica
Documentos de prueba de la aplicación	Basadas en los casos de uso	Reglas UML	Documento de pruebas de unidad	Pruebas de desarrollo
Guías para la instalación de la aplicación	Proporciona guías para elaborar manual	Diagramas UML	Diagrama	Documento técnico

Tabla XIII. Matriz de comparación de componentes utilizados en las metodologías de desarrollo

La comparación que se realizo en la tabla XIII, hace referencia a los componentes o herramientas que sirven como soporte a la consecución de los objetivos o metas que se hayan trazado en cada metodología de desarrollo.

Esta comparación se centra en reconocer cuales son las herramientas que se utilizan durante todo el ciclo de desarrollo de software, que se unen para formar una metodología de desarrollo.

5.3. Comparación de productos generados por fase

En cada fase de las metodologías de desarrollo de software se deben obtener resultados que se representan en la mayoría de los casos en documentos, diagramas o manuales. Esta representación de los resultados de cada fase de la metodología de desarrollo propuesta, la he denominado como producto.

En esta sección se presenta una comparación entre los productos generados por cada fase en la metodología de desarrollo propuesta y los productos generados en otras metodologías de desarrollo en cada una de sus fases.

En el apartado "5.3.1. Productos comunes" se presenta una descripción de los productos que se generan de manera común entre la metodología de desarrollo propuesta y otras metodologías de desarrollo.

5.3.1. Productos comunes

La siguiente lista muestra los productos comunes generados en cada fase de las metodologías escogidas para la comparación con los productos generados en las fases de la metodología propuesta:

1. Soporte de reuniones y/o entrevistas con el usuario
2. Análisis de factibilidad
3. Mapa conceptual preliminar de la aplicación
4. Mapa detallado del diseño de la aplicación

5. Soporte de validación del diseño de la aplicación
6. Archivos de código fuente documentado
7. Soporte de pruebas de la aplicación
8. Manual de instalación de la aplicación
9. Manual de usuario de la aplicación
10. Manual técnico de la aplicación

5.3.2. Matriz de comparación de productos generados por fase

En la tabla XIV se puede observar la comparación entre los productos generados en las fases de la metodología de desarrollo propuesta y los productos generados en cada fase de las metodologías orientadas a objetos y metodologías agiles.

Productos generados	Metodología Propuesta	Rational Unified Process	Object-oriented software process (OOSP)	Xtreme Programing
Soporte de reuniones y/o entrevistas con el usuario	Análisis	Iniciación	Análisis y abstracción	Exploración
Análisis de factibilidad	Análisis	--	--	--
Mapa conceptual preliminar de la aplicación	Modelado conceptual	Elaboración	Modelado conceptual	--
Mapa detallado del diseño de la aplicación	Modelado Conceptual	Elaboración	Modelado conceptual	Planning y diseño simple
Soporte de validación del diseño de la aplicación	Modelado Conceptual	--	--	--
Archivos de código fuente documentado	Elaboración de componentes	Construcción	Programación	Programación
Soporte de pruebas de la aplicación	Elaboración de componentes – Pruebas e implementación	Construcción	Programación	--
Manual de instalación de la aplicación	Elaboración de componentes	Transición	Instalación	Implementación
Manual de usuario de la aplicación	Pruebas e implementación	Transición	Instalación	Implementación
Manual técnico de la aplicación	Elaboración de componentes	Transición	Instalación	Implementación

Tabla XIV. Matriz de comparación de productos generados por fase en las metodologías de desarrollo

Es interesante mencionar finalmente una comparación de cuáles son los productos generados en cada una de las fases correspondientes a las metodologías de desarrollo que intervienen en la presente comparación.

Esta matriz de comparación se encuentra formada de la siguiente manera, en primer lugar se presentan los productos de salida común de cada fase entre las metodologías de desarrollo seleccionadas y la propuesta. Luego se presenta la descripción de la fase perteneciente a una metodología en particular, indicando las descripciones equivalentes de las fases para aquellas que son distintas en descripción pero similares en el contenido de las mismas.

CAPÍTULO VI

"APLICACIÓN DE LA METODOLOGÍA DE DESARROLLO A UN PROTOTIPO DE SOFTWARE"

CAPÍTULO VI

6. APLICACIÓN DE LA METODOLOGÍA DE DESARROLLO A UN PROTOTIPO DE SOFTWARE

Los prototipos son cruciales para diseñar un buen sitio web, facilitan la planificación del proceso de creación, reducen el coste de las evaluaciones, aumentan su efectividad y evitan graves errores en el diseño.

6.1. Objetivo y alcance del prototipo

El objetivo del prototipo es mostrar un ejemplo de la aplicación de la metodología de desarrollo de software propuesta.

El prototipo consiste en el desarrollo de la aplicación para la gestión de los documentos de Providencias Judiciales, para el departamento de Secretaría General de La Superintendencia de Bancos y Seguros.

6.2. Aplicación de la metodología

La aplicación de Providencias Judiciales se desarrolló bajo el lenguaje de programación Java, utilizando una base de datos Oracle. Cabe citar que el equipo de desarrollo estaba limitado a una persona, por lo que la solución adoptada puede no ser completamente eficiente para un ambiente de producción. Se ha dividido el desarrollo del prototipo en tareas ejecutadas por cada fase de metodología de desarrollo y una sección especial para la definición del Mapa estático del Negocio y condicionamiento de los objetivos.

6.2.1. Evaluar las condiciones del desarrollo del prototipo

En esta sección del documento se formula una definición de las condiciones endógenas y exógenas, que se tuvieron al momento del

desarrollo del presente prototipo de software. Con la definición obtenida se realiza una pequeña evaluación de los riesgos, tomando medidas preventivas.

6.2.1.1. Condiciones endógenas

Las condiciones endógenas en el desarrollo de este prototipo, se consideran de tipo normal, es decir, que los requisitos técnicos internos (plataforma de sistema operativo, herramientas de desarrollo, herramientas de ofimática, herramientas para elaboración de diagramas) estuvieron disponibles durante el desarrollo del prototipo. También se dispuso del apoyo logístico de las autoridades de la SBS, para permitir el acceso físico a las áreas de la SBS, involucradas en el desarrollo del prototipo.

6.2.1.2. Condiciones exógenas

Las condiciones de tipo exógenas, sí generaron un alto riesgo con respecto al cambio de normativas que se dio durante el proceso de construcción del prototipo.

Al inicio del desarrollo del prototipo se tomaron medidas preventivas, debido a que al tratarse la Superintendencia de Bancos y Seguros un organismo público, se pudo prever en forma temprana, las disposiciones que podrían venir en cualquier momento desde la Presidencia de la República. Las medidas que se tomaron se refieren a realizar un desarrollo parametrizable en la medida de lo posible, teniendo en consideración durante el desarrollo, que las funcionalidades de la nueva aplicación Web se encontrarán dentro del marco de las políticas actuales de gobierno.

6.2.2. Especificaciones del desarrollo del prototipo por cada fase de la metodología

En esta sección del documento se especifican las tareas que se realizaron para el desarrollo del prototipo de la aplicación de Providencias Judiciales, logrando como producto final una versión 1.0 de la aplicación Web, la misma que se puso en funcionamiento desde agosto de 2008 en el servidor de aplicaciones de la SBS.

A continuación se presenta cuáles fueron las tareas realizadas en cada una de las fases de la metodología de desarrollo propuesta, para la elaboración del prototipo.

6.2.2.1. Definición del Mapa estático del Negocio

Con la colaboración de la Subdirección de Desarrollo Institucional de la SBS se realizó la elaboración del cuadro del Mapa Estático del Negocio, el mismo que se muestra a continuación:

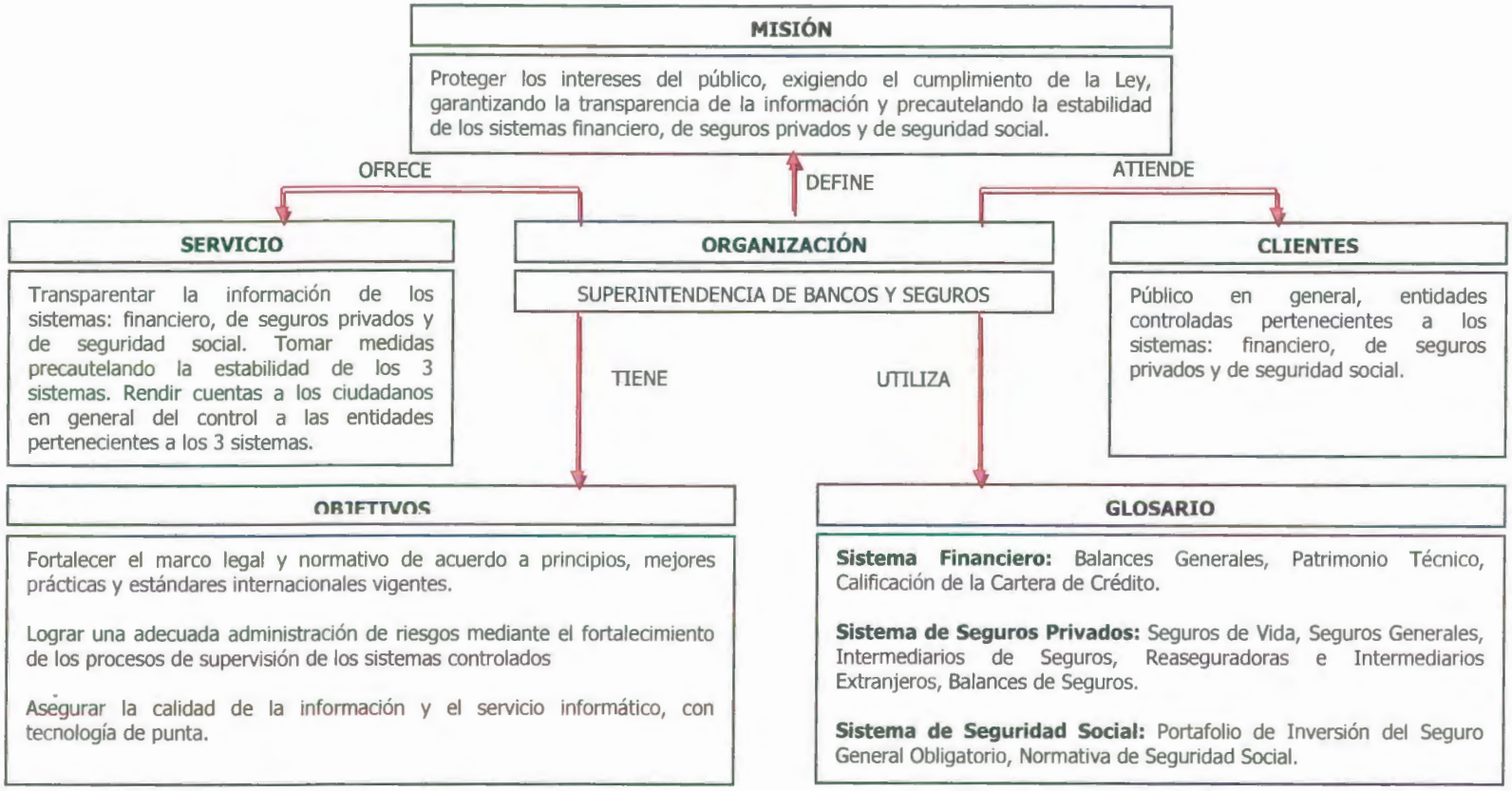


Figura 6-1 Mapa Estático del Negocio de la SBS

6.2.2.2. Tareas realizadas en la fase "Análisis de requerimientos"

En la fase de Análisis de requerimientos, por tratarse de un proyecto piloto la aplicación de Providencias Judiciales, se procedió a realizar directamente la redacción del documento de especificación de requisitos, teniendo como soporte una grabación de audio de los requerimientos solicitados por el usuario. También se hizo uso de un formulario de bitácoras de reuniones para señalar los puntos más importantes a ser tomados en consideración al momento de la elaboración del prototipo.

6.2.2.2.1. Tareas elaboradas en la sub-fase "Investigación de requisitos"

En esta sub-fase se realizó una grabación de los requerimientos del usuario en un archivo de audio, con la debida autorización de las personas involucradas, pero con la condición de que la grabación sea sólo para uso interno de la SBS.

Para los registros de la elaboración de esta tesis se utilizó un formulario de bitácoras de reuniones, el mismo que puede ser observado en el "ANEXO-11-Formulario de bitácora de reuniones del prototipo de software".

Otro producto que se generó en esta sub-fase fue el documento de especificación de requisitos del software (DRS). En el formato propuesto del DRS, al tratarse de un prototipo de la aplicación de Providencias Judiciales, se utilizó sólo los puntos más relevantes del formato de dicho documento. El Documento de Requisitos de Software (DRS), correspondiente a la nueva aplicación Web se encuentra en el "ANEXO-12-Documento de Requisitos de Software del prototipo".

6.2.2.2.2. Tareas elaboradas en la sub-fase "Análisis de requisitos"

En esta sub-fase se elaboró el documento de estudio de factibilidad de la aplicación de Providencias Judiciales. Para

redactar este documento se utilizaron los documentos de los anexos:

- ANEXO-11-Formulario de bitácora de reuniones del prototipo de software.
- ANEXO-12-Documento de Requisitos de Software del prototipo.

El documento de estudio de factibilidad se encuentra en el "ANEXO-13- Documento de Estudio de Factibilidad del prototipo".

6.2.2.3. Tareas realizadas en la fase "Modelado Conceptual"

En la fase de Modelado Conceptual se realizó el diagrama de clases de acuerdo a las especificaciones de los requisitos funcionales que se encuentran en el ANEXO-12-Documento de Requisitos de Software del prototipo.

También se elaboraron los siguientes diagramas:

- De interacción o colaboración.
- Entidad/Relación.

Los diagramas se encuentran en el "ANEXO-14-Diagramas para el desarrollo del prototipo".

6.2.2.4. Tareas realizadas en la fase "Elaboración de Componentes"

En la fase de elaboración de componentes se autogeneró código con la herramienta Oracle JDeveloper. Luego se agregó funcionalidad a los métodos más importantes de las clases.

En esta fase se elaboró el manual de instalación o despliegue, para el posterior soporte de los administradores de los servidores de aplicaciones de la SBS. El manual de despliegue se encuentra en el "ANEXO-8-Estándares de instalación y despliegue".

También se realizó el manual de usuario de la aplicación, que puede ser observado en el "ANEXO-9-Guía para realizar el manual de usuario de las aplicaciones".

6.2.2.5. Tareas realizadas en la fase "Pruebas e Implementación"

En esta fase se realizó pruebas de la aplicación junto al usuario con la descripción de los casos de uso. Por tratarse de un prototipo no se redactaron los casos de uso de prueba, que no son otra cosa que asignarles valores de ejecución a las variables encontradas en los casos de uso originales.

Finalmente luego del éxito en la aceptación por parte del usuario del nuevo prototipo, se procedió a darle más funcionalidades, convirtiéndolo en una aplicación más madura.

Entonces se decide realizar una instalación de la primera versión de la aplicación de Providencias Judiciales en el servidor de aplicaciones, para lo cual se utilizó el manual de despliegue del "ANEXO-8-Estándares de instalación y despliegue".

6.3. Resultados de la metodología

La aplicación de la metodología de desarrollo se la hizo tratando de aplicar todos los puntos de ella, incluyendo además la mayoría de los productos (documentos, diagramas, manuales) que se generan en cada fase.

A continuación se realiza una clasificación de los resultados obtenidos en cada fase de la metodología, incluyendo una primera sección donde se realizó la definición del Mapa estático del Negocio de la SBS.

En la sección de **Mapa estático del Negocio** se desarrollaron los cuadros de objetivos y estrategias actuales de la Superintendencia de

Bancos y Seguros, y se deja además establecida la recomendación de que estos cuadros deberán ser actualizados cuando se realicen cambios en los objetivos y estrategia de la SBS.

En la sub-fase **Investigación de Requisitos** se realizó la tarea de armar todos los bosquejos de los documentos en formato Microsoft Word. Dicha documentación se compone de los documentos generados en base a las plantillas definidas según la metodología propuesta, es decir, el documento de especificación de requisitos, los formularios de bitácoras de reuniones y los soportes de las entrevistas.

En la sub-fase de **Análisis de Requisitos** se realizó la elaboración del documento de estudio de factibilidad, basado en los documentos generados en la sub-fase de investigación de requisitos.

En la fase de **Modelado Conceptual** se elaboró el diagrama de clases, diagrama Entidad/Relación y el diagrama de interacción (colaboración)

con las operaciones más importantes de la aplicación de Providencias Judiciales.

En la fase de **Elaboración de Componentes** se realizó la codificación de los objetos o clases principales para el funcionamiento del proceso de publicación de archivos electrónicos. En esta fase se generó el manual de usuario y manual de instalación o despliegue.

En la fase de **Pruebas e implementación** se realizaron pruebas en conjunto con el usuario, utilizando los casos de uso. También se realizó un seguimiento en la instalación del prototipo de la aplicación en los servidores de aplicaciones de la SBS, basados en el manual de instalación o despliegue elaborado en la fase de Elaboración de Componentes.

El tiempo aproximado del desarrollo del prototipo de la aplicación Web "Providencias Judiciales" fue de 16 días. Si no se hubieran aplicado las recomendaciones de la metodología de desarrollo propuesta, el tiempo

estimado de desarrollo del prototipo es de 10 días. Cabe señalar que el prototipo de la aplicación Web desarrollado sin seguir las recomendaciones de la metodología de desarrollo, tendría problemas de repudio del usuario, no habría orden de los documentos del proyecto, sería difícil el desarrollo de la aplicación final y por consiguiente se mantendrían todos los problemas de no utilizar una metodología de trabajo para desarrollar software.

Finalmente el gran resultado de la aplicación de la metodología de desarrollo propuesta, se dio al obtener una primera versión de la aplicación Web de notificación electrónica de los documentos de Providencias Judiciales, a las entidades del sistema controlado por la SBS.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES Y RECOMENDACIONES

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software.

Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.

Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes.

La metodología de desarrollo propuesta ofrece una solución casi a medida para una gran cantidad de proyectos que tienen estas características, sin olvidar el requisito de respaldarse en la documentación por parte de la SBS.

Una de las cualidades más destacables de esta metodología es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo.

Sin embargo, hay que tener presente una serie de inconvenientes y restricciones para su aplicación, tales como:

El entorno físico debe ser un ambiente que permita la comunicación y colaboración entre todos los miembros del equipo durante todo el tiempo, cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios puede llevar al proceso al fracaso (el clima de trabajo, la colaboración y la relación interpersonal son claves), etc.

Esta metodología trata de adaptarse al contexto del proyecto, utilizando herramientas que permiten mejorar la calidad del desarrollo de algún software. De esta manera, la metodología de desarrollo propuesta intenta abordar la mayor cantidad de las situaciones contextuales del proyecto,

exigiendo un esfuerzo considerable de sus desarrolladores para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes.

Para la demostración de la eficiencia y verificación del desarrollo de software de mayor nivel de calidad, se desarrolló un sistema de aplicación práctica con un lenguaje de programación visual, donde, se aplicó la metodología de desarrollo propuesta, obteniendo como resultado un producto de calidad.

En el desarrollo y pruebas del desarrollo de sistemas basados en las guías proporcionadas por la metodología de desarrollo propuesta, se pudo llegar a varias conclusiones.

Podemos decir que las guías y directrices utilizadas fueron las adecuadas, pues se logró construir herramientas que cumplieran con los propósitos propuestos inicialmente.

Asimismo esta metodología de desarrollo, puede mejorar el aprendizaje de las partes principales de las metodologías de Ingeniería de software más conocidas para el desarrollo de software.

Otra característica relevante de esta metodología de desarrollo es que al utilizarse el método de Análisis Estructurado (profundidad de la fase de análisis), se puede disminuir el riesgo de la comprensión de lo que realmente debe realizar el software, y ser útil como referencia para las siguientes fases del ciclo de desarrollo de software.

Finalmente se puede concluir que el objetivo de esta tesis ha sido cumplido al darle al usuario no sólo una guía útil y que cuenta con los requerimientos necesarios para el desarrollo de un nuevo software, sino una directriz que auxilia a los técnicos del área de sistemas a realizar sus proyectos y también favorece y estimula el uso y aprendizaje de muchas tecnologías, técnicas y métodos para el desarrollo de proyectos de software.

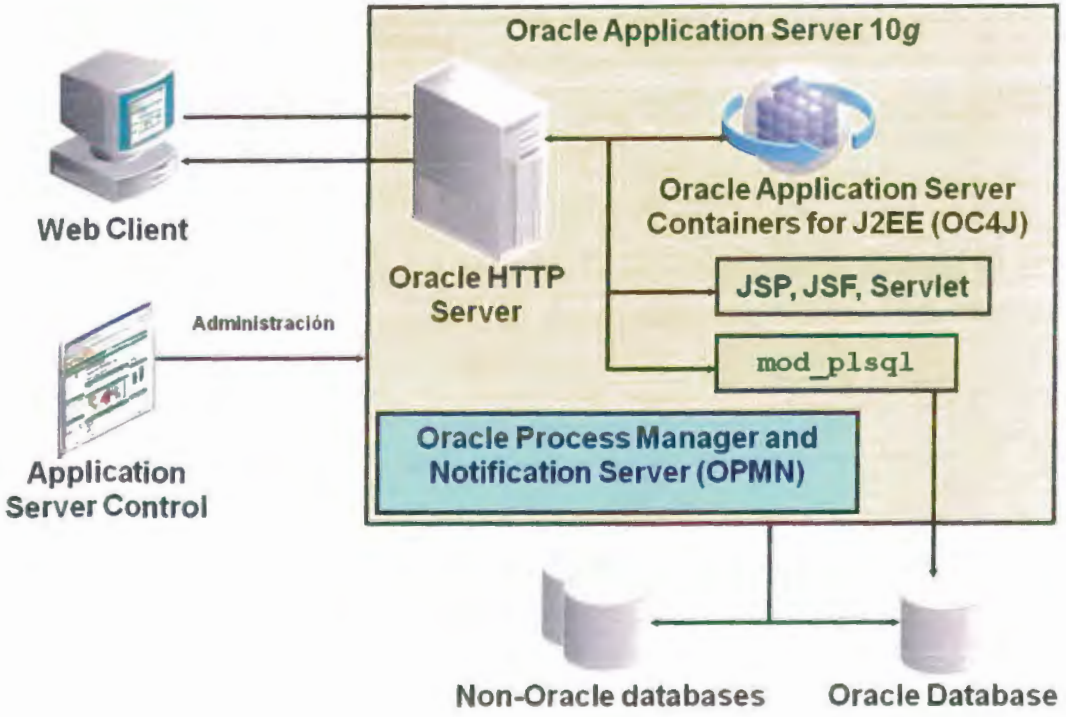

MSIG
MAESTRÍA EN SISTEMAS DE
INFORMACIÓN GERENCIAL
ESCUELA SUPERIOR POLITECNICA DEL LITORAL
ESPOL

ANEXOS

ANEXOS

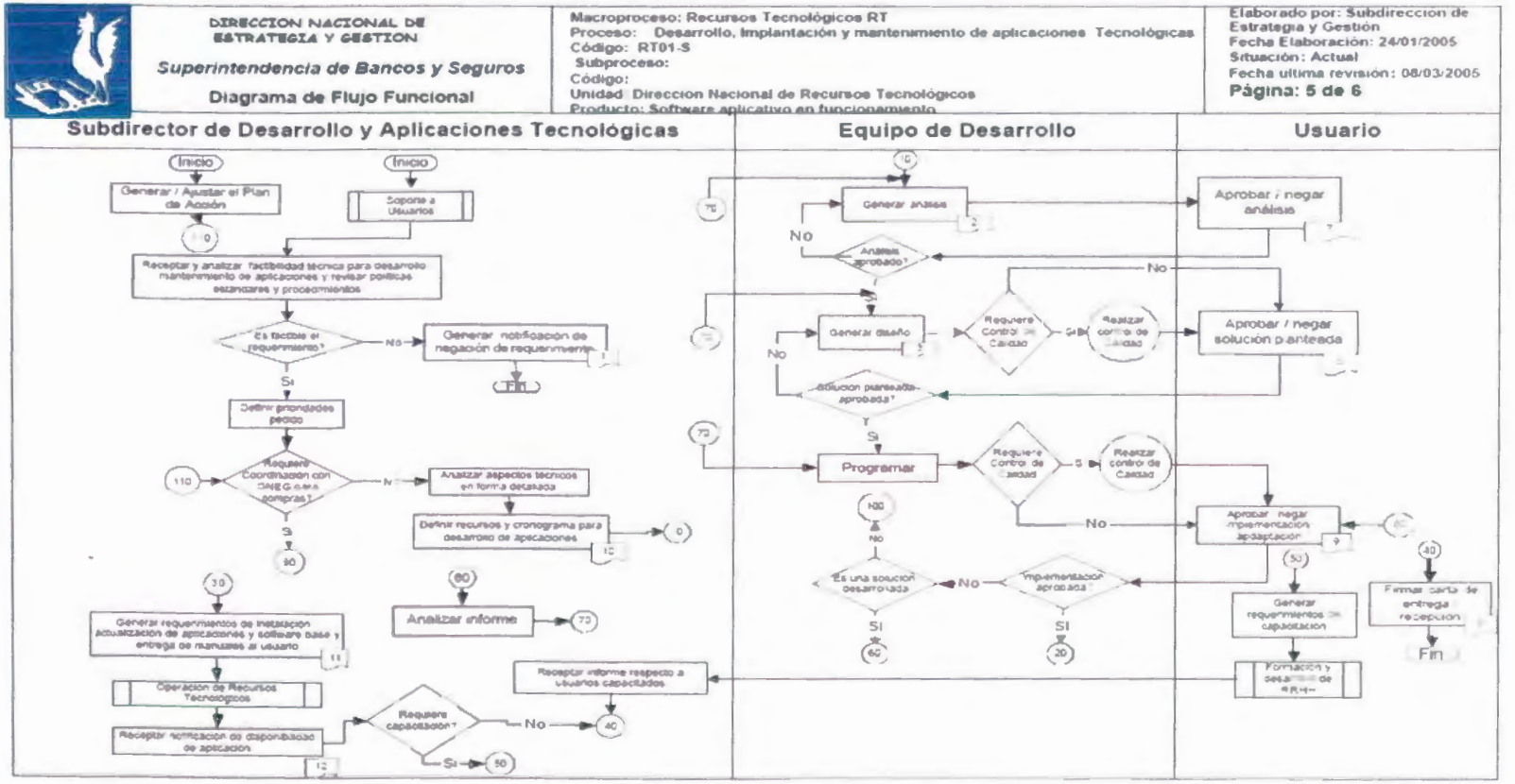
"ANEXO-1-Diagrama de la arquitectura tecnológica"

En este anexo se muestra la arquitectura tecnológica actual de la Superintendencia de Bancos y Seguros, con respecto a la distribución de las capas lógicas y físicas para el soporte de aplicaciones Web. La presente propuesta de la metodología de desarrollo por pedido de la Superintendencia de Bancos y Seguros, ha tenido que adaptarse a ella.



"ANEXO-2-Diagrama de flujo de procesos internos de desarrollo de software"

En este anexo se muestra las modificaciones realizadas al nuevo flujo de tareas encaminadas en un nuevo desarrollo de aplicaciones Web. Se hizo un mayor énfasis en las etapas de la ingeniería de requisitos y en la parte de implementación o instalación.



DIRECCION NACIONAL DE ESTRATEGIA Y GESTION

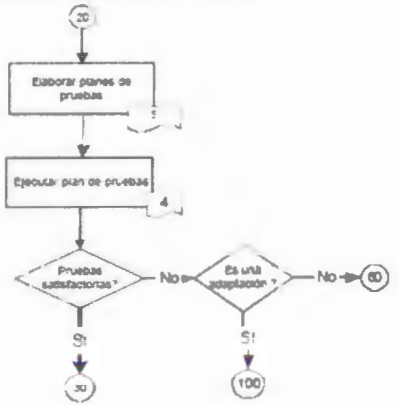
Superintendencia de Bancos y Seguros

Diagrama de Flujo Funcional

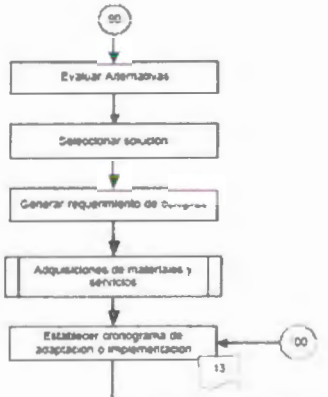
Macroproceso: Recursos Tecnológicos RT
Proceso: Desarrollo, implantación y mantenimiento de aplicaciones Tecnológicas
Código: RT01.5
Subproceso:
Código:
Unidad: Dirección Nacional de Recursos Tecnológicos
Producto: Software aplicativo en funcionamiento

Elaborado por: Subdirección de Estrategia y Gestión
Fecha Elaboración: 24/01/2005
Situación: Actual
Fecha última revisión: 08/03/2005

Equipo de Pruebas



Equipo de Selección

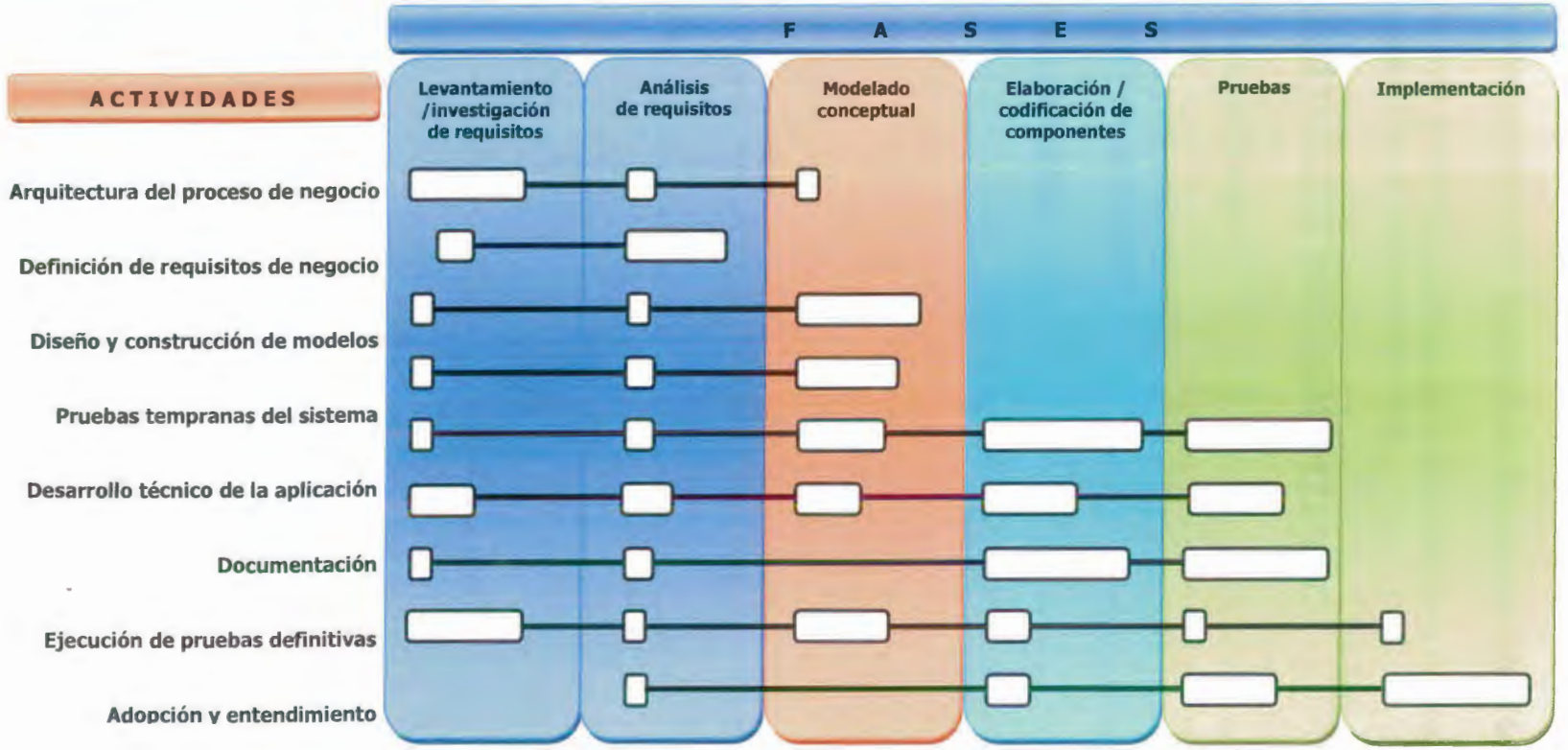


Proveedor

Documentos:
1) Notificación de negación de requerimiento a Usuario b Archivo Sistemas Informáticos
2) Documentación de análisis a Usuario b Archivo Sistemas Informáticos
3) Documentación de diseño a Usuario b Archivo Sistemas Informáticos
4) Informe de Ejecución de Pruebas a Archivo Sistemas Informáticos
5) Plan de pruebas a Equipo de Pruebas b Archivo Sistemas Informáticos
6) Acta de entrega - recepción de aplicación o ajuste a Usuario b Archivo Sistemas Informáticos
7) Acta de aprobación de análisis a Usuario b Archivo Sistemas Informáticos
8) Acta de aprobación de diseño a Usuario b Archivo Sistemas Informáticos
9) Acta de aprobación de implementación a Usuario b Archivo Sistemas Informáticos
10) Cronograma de desarrollo ajuste de aplicación a Archivo Sistemas Informáticos
11) Requerimiento de instalación actualización de aplicaciones y software base a Responsable de proceso b Sistemas Informáticos
12) Notificación de disponibilidad de aplicación a Usuario b Archivo Sistemas Informáticos
13) Documento de requerimiento de ajustes

"ANEXO-3-Diagrama de flujo de las fases de la metodología propuesta"

En este anexo se presenta el flujo de las fases de la metodología de desarrollo de aplicaciones Web propuesta. En este flujo se realiza un especial énfasis en las actividades que deben ejecutarse en una o varias de las fases de la metodología. El flujo de las fases de la metodología engrana con el flujo del proceso de desarrollo definido en la Superintendencia de Bancos y Seguros para el área de la Subdirección de Desarrollo y Aplicaciones Tecnológicas.



"ANEXO-4.1-Formulario para la solicitud de requerimientos"

En este anexo se presenta el formato de impresión del formulario de solicitud de requerimiento de desarrollo de aplicaciones Web.

SOLICITUD DE REQUERIMIENTO

Fecha de la solicitud (dd/mm/yyyy): .....
Área usuaria solicitante: .....
Nombre del titular/usuario del área solicitante: .....
Nombre de la aplicación / requerimiento: .....

Descripción del requerimiento:

.....
.....
.....
.....
.....
.....
.....

Descripción de anexos:

Tipo de documento	No. de hojas.

Anexos:

Archivos ( )

Impresiones ( )

Otros ( )
Especificar :

Atentamente,

_____

Nombre de la persona que solicita el requerimiento
Cargo.

"ANEXO-4.2-Formulario de bitácora de reuniones con el usuario"

En este anexo se presenta el formato de impresión del formulario de bitácoras de los acuerdos obtenidos en las reuniones de trabajo con el usuario.

BITACORA DE REUNION

Bitácora de reuniones en el proceso de desarrollo de aplicaciones de la Superintendencia de Bancos y Seguros, Dirección Nacional de Recursos Tecnológicos, Subdirección de Desarrollo y Aplicaciones Tecnológicas.

Lugar:	(Lugar donde se llevo a cabo la reunión).....
Fecha:	(Fecha de la reunión).....
Horario:	Duración de la reunión, hora (mm:ss) de inicio y fin.....
Proyecto:	(Nombre del proyecto al cual se esta haciendo referencia)
Etapa:	(Etapa dentro del proceso de desarrollo en la que tiene lugar esta reunión)
Tema a tratar:	.....
Asistentes:	.....

1. **Asuntos tratados en la reunión.**

   - **Agenda**
     Se hace un resumen narrativo de lo que se hablo en la reunión, se especificaran puntos tratados.

   - **Restricciones (en caso que aplique)**
     Se explicaran las posibles limitaciones, restricciones que se puedan establecer como resultado de lo que se pudo establecer en la reunión.

2. **Pendientes.**
   Se establece que quedara pendiente para la próxima reunión.

3. **Conclusiones**
   Conclusiones a la que se llega con lo que se trato en la reunión.

4. **Próxima reunión.**
   Si en esta reunión se acordó mantener una próxima reunión se podrá la fecha en que se la pacto.

5. **Acuerdos alcanzados**
   Acuerdos que llegaron entre ambas partes.

Los acuerdos alcanzados en esta reunión son la base para el desarrollo del proyecto.

Atentamente,

SDAT	ÁREA USUARIA
_____	_____
Nombre del técnico Desarrollo de aplicaciones	Nombre del usuario contraparte Área a la que pertenece.

C.I.B.

"ANEXO-4.3-Documento de estudio de factibilidad"

En este anexo se presenta el formato de impresión del documento de estudio de factibilidad para el desarrollo de una nueva aplicación Web.

ESTUDIO DE FACTIBILIDAD

Proyecto de automatización: (Nombre que el usuario da al requerimiento)

1. Identificación del área solicitante

Fecha del estudio (dd/mm/yyyy): (Fecha en que se finalizo en estudio)

Area usuaria de la aplicación: .....

Nombre del titular del área usuaria: .....

Usuario contraparte: .....

No. de horas comprometidas por el usuario contraparte: (el usuario contraparte pueden ser varios, detallar)

2. Especificaciones del Proyecto

2.1 Nombre del Sistema: .....
2.2 Siglas de identificación: .....
2.3 Versión: .....
2.4 Objetivos del proyecto
- .....

2.5 Principales problemas a resolver.
- .....

2.6 Restricciones, limitantes
- .....

2.7 Beneficios que se obtendrán
- .....

2.8 Usuarios principales y roles que cumplen actualmente
- .....

2.9 Infraestructura de hardware y software necesaria para la solución
- .....

2.10 Tecnologías de desarrollo

2.11 Tiempo aproximado de desarrollo

(Se adjunta plan de trabajo detallado)

2.12 Presupuesto Referencial

Se calcula en función del costo por remuneraciones del personal involucrado y el tiempo estimado de desarrollo.

3. Descripción conceptual de la aplicación

Se hace un resumen narrativo y/o gráfico de la forma en que funcionaría la aplicación, del alcance y las limitaciones que deben considerarse y de los riesgos que corre la aplicación en el caso de producirse eventos no previstos como cambios de estructura, incumplimiento de los procedimientos acordados, retiro del personal contraparte, etc.

4. Conclusión sobre la factibilidad del proyecto

Se manifiesta un criterio técnico sobre la factibilidad de emprender el proyecto en las condiciones de funcionamiento y recursos actuales.
De requerirse una cantidad de recursos de personal, hardware o software adicionales, para el desarrollo o el futuro funcionamiento de la aplicación, el criterio sobre la factibilidad técnica estaría condicionado a la aceptación de la autoridad para facilitar esos recursos.

Manifestamos nuestra conformidad con lo arriba expuesto.

Atentamente,

_____	_____
Nombre del titular del Área Solicitante Cargo	Usuario Contraparte Área
	_____
	Usuario Contraparte Área
_____	_____
Nombre Subgerente de Software Aplicativo Cargo	Informático asignado Desarrollo de Aplicaciones

"ANEXO-5-Estándares de codificación-Java"

A5.1. Introducción

A5.1.1. Generalidades

En esta sección se indican los estándares de codificación propuestos para la metodología de desarrollo.

Los estándares de codificación se especifican sobre los lenguajes Java, XML, sobre la organización de archivos.

A5.2. Nombres de archivos

Esta sección lista los sufijos y nombres de los archivos más comúnmente usados.

A5.2.1. Sufijos de archivos

Para archivos con código Java dentro existen los siguientes sufijos:

Tipo de archivo	Sufijo
Fuente Java	.java
ByteCode Java	.class

A5.2.2. Nombres de archivos comunes

Los nombres de los archivos frecuentemente usados incluyen:

Nombre de archivo	Uso
GNUmakefile	El nombre de preferencia para makefiles. Nosotros usamos gnumake para construir nuestro software.
Leeme	El nombre de preferencia para el archivo que resume el contenido de un directorio específico.

A5.3. Organización de archivos

Un archivo consiste de secciones que deberían ser separadas por líneas en blanco y un comentario opcional, identificando cada sección.

Los archivos con más de 2000 líneas son cumbersome y deberían ser avoided.

Para ver un ejemplo de un fuente de programa codificado en lenguaje Java, revisar el anexo x "Java Source File Example".

A5.3.1. Archivos fuente en lenguaje Java

Cada archivo fuente codificado en lenguaje Java contiene una sola declaración de la sintaxis *public class* o *interface*. Cuando las clases de privadas (private) y las interfaces son asociadas en una *public class*, se las puede poner dentro de un mismo archivo fuente en Java como una *public class*. La sintaxis *public class* debería estar al inicio de la clase o interfaz en el archivo.

Los archivos fuentes en Java tienen el siguiente ordenamiento:

Comentarios iniciales (ver "3.1.1. Comentarios Iniciales")
Sentencias Packages e Imports; por ejemplo:

```
import java.util.*;
import java.lang.*;
import java.net.*;
```

Declaraciones Class e interface (ver "3.1.3. Declaraciones Class e Interface")

A5.3.1.1. Comentarios Iniciales

Todos los archivos fuente en Java deben comenzar con un comentario que indique lo siguiente: la lista de nombres de programadores, la fecha de creación y modificación con el autor de los cambios, el comentario de los derechos reservados, y lo más importante una breve descripción del propósito del programa Java. Por ejemplo:

```
/*
* Classname
*
* Version 1.0
*
* Copyright 2008 Superintendencia de Bancos y Seguros
*/
```

A5.3.1.2. Sentencias *Package* e *Import*

La primera línea que no significa un comentario en muchos archivos fuentes en lenguaje Java es una sentencia *package statement*. Después de esta sentencia puede continuar el uso de una o más sentencias *import*. Por ejemplo:

```
package java.awt;
import java.awt.peer.CanvasPeer;
```

A5.3.1.3. Declaraciones *Class* e *Interface*

La siguiente tabla describe las partes en la declaración de una clase (class) o de una interfaz (interface), especificando el orden en el que deberían aparecer dentro del archivo fuente. Ver el anexo x "Java Source File Example", en este ejemplo se incluyen líneas de comentarios.

Orden	Partes en la declaración de Class/Interface	Notas
1	Comentario de tipo documentación de la `Class/interface (/**...*/)`	Ver "5.2. Comentarios para documentación final" para más información de que se debería poner en este tipo de comentario.
2	Sentencia `class` or `interface`	
3	Comentario de tipo implementación de la `Class/interface (/*...*/)`, si es necesario	Este comentario debería contener alguna información de alguna implementación en particular en la Class/interface. Se debe diferenciar del comentario de tipo documentación.
4	Variables `Class (static)`	Primero se deben declarar las variables publicas de clase `(public class variables)`, luego las protegidas `(protected)`, y después las privadas `(private)`.
5	Variables de instancia	Primero se deben declarar las variables `public`, luego las `protected`, y después las `private`.
6	Constructores (Constructors)	

7	Metodos (Methods)	Estos métodos deben ser agrupados por funcionalidad, dependiendo de su alcance (scope) o accesibilidad(accessibility) Por ejemplo, un método privado de clase puede estar entre dos métodos públicos de instancia. El objetivo es hacer de fácil lectura y de fácil comprensión el código.

A5.4. Identación

Se deben utilizar cuatro espacios como unidad de identación. La construcción exacta de la identación (spaces vs. tabs) no se puede especificar. Los Tabs equivalen a una identación de ocho espacios (no de cuatro espacios)

A5.4.1. Longitud de la línea

No se deben crear líneas de más de 80 caracteres, debido a la dificultad que presentan ciertas herramientas para manipular líneas muy grandes en caracteres.

Nota: Los bloques que contienen alguna documentación deben tener una longitud de línea corta, es decir, hasta 70 caracteres como máximo.

A5.4.2. Wrapping Lines

Cuando una expresión no cabe en una sola línea de código, la expresión debe dividirse en varias líneas de acuerdo a estos principios generales:

- Dividir la expresión después de una coma.
- Antes de un operador.
- Es preferible mantener un nivel de caracteres alto para dividir una expresión en varias líneas a tener una división con pocos caracteres por cada línea.
- Alinear los caracteres de la nueva línea, al mismo nivel del comienzo de la expresión en la línea anterior.
- Si las reglas propuestas hacen muy confuso el código o el código está muy disperso, llevar las líneas de código al margen derecho y solo identarlas con ocho espacios.

Algunos ejemplos de quiebres de línea de código en llamadas a métodos:

```
function(longExpression1, longExpression2, longExpression3,
        longExpression4, longExpression5);
```

```
var = function1(longExpression1,
                function2(longExpression2,
                          longExpression3));
```

Los siguientes dos ejemplos se refieren a los quiebres de línea de código de una expresión aritmética. El primer ejemplo es el recomendado, debido a que se mantiene una mayor longitud de caracteres por línea antes de generar la siguiente línea con la continuación de la expresión. El segundo ejemplo utiliza menos caracteres en la primera línea de código lo cual no es recomendable ya que el código puede crecer mucho y resultar confuso:

```
longName1 = longName2 * (longName3 + longName4 - longName5)
            + 4 * longname6; // PREFER

longName1 = longName2 * (longName3 + longName4
                         - longName5) + 4 * longname6; // AVOID
```

Los siguientes dos ejemplos son de identación en la declaración de métodos. El primero trata del caso convencional. El segundo ejemplo elevaría el uso de líneas y de caracteres por línea si se utiliza la identación convencional, es por esta razón que es recomendable utilizar la regla de identación de solo ocho espacios en cada nueva línea de código:

```
//CONVENTIONAL INDENTATION
someMethod(int anArg, Object anotherArg, String yetAnotherArg,
           Object andStillAnother) {
    ...
}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS
private static synchronized horkingLongMethodName(int anArg,
        Object anotherArg, String yetAnotherArg,
        Object andStillAnother) {
    ...
}
```

Line wrapping para sentencias if deben generalmente usar la regla de ocho espacios de identación y no la convencional de 4 espacios, ya que esta ultima hace que el código del cuerpo del programa se visualice de una manera dificultosa. Por ejemplo:

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) { //BAD WRAPS
    doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS
}

//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
        || (condition3 && condition4)
```

```
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}

//OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
        ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```

A continuación se presentan tres maneras aceptables para el formato de ternary expressions:

```
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
        ? beta
        : gamma;
```

A5.5. Comentarios

Hay dos tipos de comentarios en los programas Java: comentarios de implementación y comentarios para la documentación. Los comentarios de implementación son los que se utilizan en el lenguaje C++, y están delimitados por /*...*/, y //. Los comentarios para la documentación (conocidos como "doc comments") son utilizados solo para el lenguaje Java, y están delimitados por /**...*/. Los comentarios para documentación se pueden ser extraer para la generación de archivos HTML, utilizando la herramienta javadoc. Los comentarios nunca deben contener caracteres especiales tales como un enter o un backspace.

A5.5.1. Formatos de comentarios de implementación

En los programas se pueden utilizar cuatro estilos de comentarios de implementación: de bloque, de una sola línea, de trailing y al final de una línea.

A5.5.1.1. Comentarios de bloque

Los comentarios de bloque son utilizados para proveer descripción de archivos, métodos, estructuras de datos y algoritmos. Los comentarios de bloque deben ser utilizados al inicio de cada archivo y antes de cada método. Estos comentarios también pueden ser utilizados en otros lugares, como por ejemplo dentro del cuerpo del método. Los comentarios de bloque internos

en una función o método deben ser identados al mismo nivel del código que se está describiendo.

Un comentario de bloque debe estar precedido de una línea en blanco para separarlo del resto del código. Los comentarios de bloque tienen un asterisco "*" al inicio de cada línea a excepción de la primera.

```
/*
 * Here is a block comment.
 */
```

Los comentarios de bloque pueden empezar con /*-, lo cual es conocido como **identación(1)** al comienzo de un comentario de bloque. Ejemplo:

```
/*
 * Here is a block comment with some very special
 * formatting that I want indent(1) to ignore.
 *
 * one
 * two
 * three
 */
```

A5.5.1.2. Comentarios de una sola línea

Son comentarios cortos que pueden aparecer en una sola línea, identado al mismo nivel del código que sigue o continua en la siguiente línea. Si un comentario no puede ser escrito en una sola línea para este formato, entonces se debería utilizar el formato de comentarios de bloque (ver sección A5.3.1.1.). Los comentarios en el formato de una sola línea deben ir precedidos de una línea en blanco. A continuación se muestra un comentario dispuesto en el formato de una sola línea:

```
if (condition) {
    /* Handle the condition. */
    ...
}
```

A5.5.1.3. Comentarios de trailing

Son comentarios muy cortos, que pueden aparecer en la misma linea del codigo al cual describe, pero que deberia estar bien separados de las sentencias. Si más de un comentario corto llegara a aparecer en una misma porción de código, entonces los comentarios deben ser identados al mismo nivel de tabs utilizados. No se debe especificar un comentario de trailing por cada línea de código ejecutable como se lo realizaba en los lenguajes ensambladores. Ejemplo:

```
if (a == 2) {
    return TRUE;          /* special case */
} else {
    return isprime(a);    /* works only for odd a */
}
```

A5.5.1.4. Comentarios al final de líneas

El comentario de final de línea es el que comienza con los caracteres "//" indicando que continua una nueva línea. Este formato no debería ser usado en multiples líneas consecutivas para realizar comentarios; sin embargo, los comentarios de final de línea pueden ser usados en múltiples líneas consecutivas para comentar ciertas secciones de código correspondientes. Ejemplos:

```
if (foo > 1) {
    // Do a double-flip.
    ...
}
else
    return false; // Explain why here.

//if (bar > 1) {
//
// // Do a triple-flip.
// ...
//}
//else
// return false;
```

A5.5.2. Comentarios para documentación

Los comentarios de documentación describen clases, interfaces, constructors, metodos, y campos escritos en lenguaje Java. Cada comentario de documentación esta dentro de los delimitadores "/**...*/", y debe existir un comentario por cada API. Estos comentarios deben aparecer justo antes de la declaración:

```
/**
 * The Example class provides ...
 */
class Example { ...
```

Hay que observar en el ejemplo anterior que las clases y las interfaces no se identan, mientras que sus miembros sí. La primera línea del comentario de documentación (/**) para clases e interfaces no se identa, y las siguientes líneas del mismo comentario tienen un espacio de identacion cada una. Los

miembros de clases e interfaces, incluyendo los constructores, tienen cuatro espacios para la primera línea de comentario de documentación y cinco espacios después de ella.

A5.6. Declaraciones

A5.6.1. Por el número de líneas

Una declaración por línea es recomendada siempre y cuando exista un comentario a continuación, por ejemplo:

```
int level; // nivel de identación
int size; // tamaño de tabla
```

Pero es preferible hacer lo declaración en una línea, por ejemplo:

```
int level, size;
```

En ningún caso se debe hacer una declaración de variables y de métodos en la misma línea. Ejemplo:

```
long dbaddr, getDbaddr(); // Incorrecto!
```

No se deben declarar diferentes tipos de datos en la misma línea. Ejen

```
int foo, fooarray[]; // Incorrecto!
```

Nota: En los ejemplos anteriores se utiliza un espacio en blanco entre el tipo de dato y el identificador. Otra alternativa es la de utilizar tabs, ejemplo:

```
int         level; // nivel de identación
int         size; // tamaño de la tabla
```

A5.6.2. Ubicación de las declaraciones

Se deben ubicar a las declaraciones sólo al comienzo de los bloques de código (un bloque es cualquier código encerrado entre llaves "{}"). No se debe esperar declarar variables hasta su primera utilización en el código; esto puede confundir al programador y a su vez causar que se disminuya la portabilidad del código con respecto al alcance de las variables. Ejemplo:

```
void MyMethod() {
      int int1; // beginning of method block
      if (condition) {
            int int2; // beginning of "if" block
            ...
```

```
        }
    }
```

Una excepción a esta regla es la declaración de índices para las sentencias "for", ejemplo:

```
for (int i = 0; i < maxLoops; i++) { ...
```

A5.6.3. Inicialización

Tratar de inicializar las variables en el lugar de su declaración. La única razón para no inicializar una variable en el lugar de su declaración, es que el valor inicial dependa de que se realice primero un cómputo o un cálculo inicial.

A5.6.4. Declaración de Clases e Interfaces

Cuando se codifican clases o interfaces, se deberían seguir las siguientes reglas de formato:

- No colocar espacios entre el nombre del método y el paréntesis "(" que indica el inicio de la lista de parámetros, si los tuviere.

- La llave de inicio "{" debe aparecer al final de la misma línea de la sentencia de declaración.

- La llave de cierre "}" debe aparecer en otra línea con un nivel de identación, que haga referencia al inicio del código donde se hizo la apertura de la llave de inicio, a excepción de que sea una declaración de sentencia nula, entonces la llave de cierre "}" debe aparecer luego de la llave de inicio "{".

```
class Sample extends Object {
        int ivar1;
        int ivar2;
        Sample(int i, int j) {
                ivar1 = i;
                ivar2 = j;
        }
        int emptyMethod() {}
        ...
}
```

- Los métodos se deben separar con una línea en blanco.

A5.7. Sentencias

A5.7.1. Sentencias Simples

Cada línea de código no debe contener más de una línea de código. Ejemplo:

```
argv++; argc--;
```

A5.7.2. Sentencias Compuestas

Las sentencias compuestas son aquellas que contienen una lista de sentencias encerradas con llaves "{}". Se deben seguir las siguientes sugerencias:

- Las sentencias dentro de las llaves "{}" deberían estar identadas un nivel más al inicio de la sentencia compuesta.
- La llave de inicio "{" debe colocarse al final de la línea de inicio de la sentencia compuesta; la llave de cierre "}" debe aparecer al inicio de una línea y estar identada al mismo nivel del inicio de la sentencia compuesta.

A5.7.3. Sentencias if, if-else, if-else-if-else

Las sentencias if-else, deberían tener la siguiente forma:

```
if (condition) {
        statements;
}
if (condition) {
        statements;
} else {
        statements;
}
if (condition) {
        statements;
} else if (condition) {
        statements;
} else if (condition) {
        statements;
}
```

A5.7.4. Sentencias for

Las sentencias for, deben tener la siguiente forma:

```
for (initialization; condition; update) {
      statements;
}
```

Una sentencia for vacía debe tener la siguiente forma:

```
for (initialization; condition; update);
```

A5.7.5. Sentencias while

La sentencia while, debe tener la siguiente forma:

```
while (condition) {
      statements;
}
```

Una sentencia while vacía debe tener la siguiente forma:

```
while (condition);
```

A5.7.6. Sentencias do-while

Una sentencia do-while, debe tener la siguiente forma:

```
do {
      statements;
} while (condition);
```

A5.7.7. Sentencias switch

La sentencia switch, debe tener la siguiente forma:

```
switch (condition) {
case ABC:
      statements;
      /* falls through */
case DEF:
      statements;
      break;
case XYZ:
      statements;
      break;
default:
      statements;
      break;
}
```

Cada sentencia switch, debe tener un default case.

A5.7.8. Sentencias try-catch

Las sentencias try-catch, deben tener la siguiente forma:

```
try {
        statements;
} catch (ExceptionClass e) {
        statements;
}
```

A5.8. Espacios en blanco

A5.8.1. Líneas en blanco

Las líneas en blanco proveen facilidad de lectura a ciertas partes del código fuente.

Se deberían utilizar siempre dos líneas en blanco en las siguientes circunstancias:

- Entre secciones de un archivo fuente.
- Entre definiciones de clases e interfaces.

Se debería utilizar siempre una línea en blanco en las siguientes circunstancias:

- Entre métodos.
- Entre las variables locales dentro de un método y la primera sentencia.
- Antes de un comentario de bloque o de línea simple.
- Entre secciones de lógica dentro del método para proveer legibilidad al código fuente.

A5.8.2. Espacios en blanco

Se deberían utilizar espacios en blanco en las siguientes circunstancias:

- Una palabra clave seguida por un paréntesis. Ejemplo:

```
while (true) {
        ...
}
```

Nota: Los espacios en blanco no se deben utilizar entre un nombre de método y el paréntesis de apertura. Esto ayuda a distinguir las palabras claves de las llamadas a métodos.

- Un espacio en blanco debe aparecer después de las comas en la lista de argumentos.

- Todos los operadores binarios excepto el punto (".") deberían estar separados por espacios. Los espacios en blanco nunca deberían separar operaciones de incremento ("++") o decremento ("—") de sus operandos. Ejemplo:

```
a += c + d;
a = (a + b) / (c * d);

while (d++ = s++) {
       n++;
}
prints("size is " + foo + "\n");
```

- Las expresiones en una sentencia "for", deberían estar separadas por espacios en blanco. Ejemplo:

```
for (expr1; expr2; expr3)
```

- Los "Casts" deberían estar seguidos por un espacio en blanco. Ejemplo:

```
myMethod((byte) aNum, (Object) x);
myFunc((int) (cp + 5), ((int) (i + 3)) + 1);
```

A5.9. Estándares de nombres

Los estándares de nombres hacen que los programas sean entendibles y legibles. También pueden dar información acerca de la función de un identificador –por ejemplo, el estado de una constante, paquete o clase- para ayudar a entender más el código fuente.

Los estándares de nombres dados en esta sección son de nivel superior.

Tipo de identificador	Reglas de nombres	Ejemplos
Clases	Los nombres de las clases deberían ser sustantivos y utilizar letra capital si en la definición existe más de una palabra. Hay que tratar de mantener los nombres de las clases simples y descriptivas.	`class Usuarios;` `class DocJudiciales;`
Interfaces	Los nombres de las interfaces deben escribirse con letra capital, muy similar al formato de nombre de las clases.	`interface MenuUsuario;`
Métodos	Los nombres de métodos deberían ser verbos, iniciando la definición con letra minúscula y desde la segunda palabra utilizar letra capital.	`cerrar();` `asignarUsuario();`

Variables	Los nombres de variables deben empezar con letra minúscula y luego desde la segunda palabra con letra capital. El nombre de variable debe ser definido de acuerdo al sentido de su uso. Se pueden utilizar nombres de un solo caracter para variables de uso temporal.	`int i;`
		`char *cp;`
		`float anchoColumna;`

Constantes	Los nombres de variables de clases constantes ó de constantes ANSI, deben definirse con letra mayúscula y cada palabra debe ir separada por un underscore ("_").	`int MIN_WIDTH = 4;`
		`int MAX_WIDTH = 999;`
		`int GET_THE_CPU = 1;`

A5.10. Prácticas de programación

A5.10.1. Acceso a variables de instancia y de clase

Una variable de instancia o de clase no debe tener acceso público, sin una buena razón. A menudo se da, que las variables de instancias no necesitan de los métodos de acceso "set" y "get".

Un ejemplo del uso apropiado de variables de instancia con acceso público es cuando la clase es esencialmente una estructura de datos sin ningún comportamiento.

A5.10.2. Constantes

Las constantes numéricas no deben ser inicializadas directamente, excepto por los valores de -1, 0 y 1 que pueden tomar en las sentencias "for", es decir, como valores de contadores para el ciclo.

A5.10.3. Valores de asignación para variables

En muchas ocasiones a variables de diferentes clases, se les asigna un mismo valor en una sola sentencia. Esto resulta difícil de leer. Ejemplo:

```
fooBar.fChar = barFoo.lchar = 'c';
```

No utilizar un operador de asignación en lugares donde puede ser fácilmente confundido por un operador de igualdad. Ejemplo:

```
if (c++ = d++) {
        ...
}
```

Se debería escribir de la siguiente manera:

```
if ((c++ = d++) != 0) {
        ...
}
```

No se debe utilizar asignaciones embebidas, para tratar de mejorar el rendimiento de ejecución del código fuente. Este trabajo es del compilador, y utilizar las asignaciones embebidas raramente puedan ayudar al desempeño. Ejemplo:

```
d = (a = b + c) + r;
```

Se debería escribir de la siguiente manera:

```
a = b + c;
d = a + r;
```

A5.11. Ejemplos de código

A5.11.1. Ejemplo de un archivo fuente en Java

El siguiente ejemplo muestra como formatear un archivo fuente escrito en lenguaje Java, que contiene una clase pública. Las interfaces en este archivo son formateadas de forma similar a la clase pública.

```
/**
 * SUPERINTENDENCIA DE BANCOS Y SEGUROS
 * DEL ECUADOR
 *
```

```
 * @Author   << Nombre Apellido>>
 * @Version  << Número de versión >>
 *
 * Ambiente de Desarrollo: JDeveloper <<versión IDE>>
 *
 * Nombre del Archivo : << Descripción >>
 *
 *
 **/
package java.utils;

import java.utils.blahdy.BlahBlah;

/**
* Descripción de la Clase.
*
* @version 1.10 04 Oct 2008
* @author Nombre Apellido
*/
public class Blah extends SomeClass {
	/* Se puede ubicar un comentario de implementación de la clase. */

	/** classVar1 comentario de documentación */
	public static int classVar1;

	/**
	 * classVar2 comentario de documentación
	 * con más de una línea
	 */
	private static Object classVar2;

	/** instanceVar1 comentario de documentación */
	public Object instanceVar1;

	/** instanceVar2 comentario de documentación */
	protected int instanceVar2;

	/** instanceVar3 comentario de documentación */
	private Object[] instanceVar3;
	/**
	 * ...método Blah comentario de documentación ...
	 */
	public Blah() {
		// ...Aquí va la implementación ...
	}

	/**
	 * ...método doSomething comentario de documentación...
	 */
	public void doSomething() {
		// ... Aquí va la implementación...
	}

	/**
	 * ...método doSomethingElse comentario de documentación...
	 * @param someParam descripción
	 */
	public void doSomethingElse(Object someParam) {
		// ... Aquí va la implementación...
	}
```

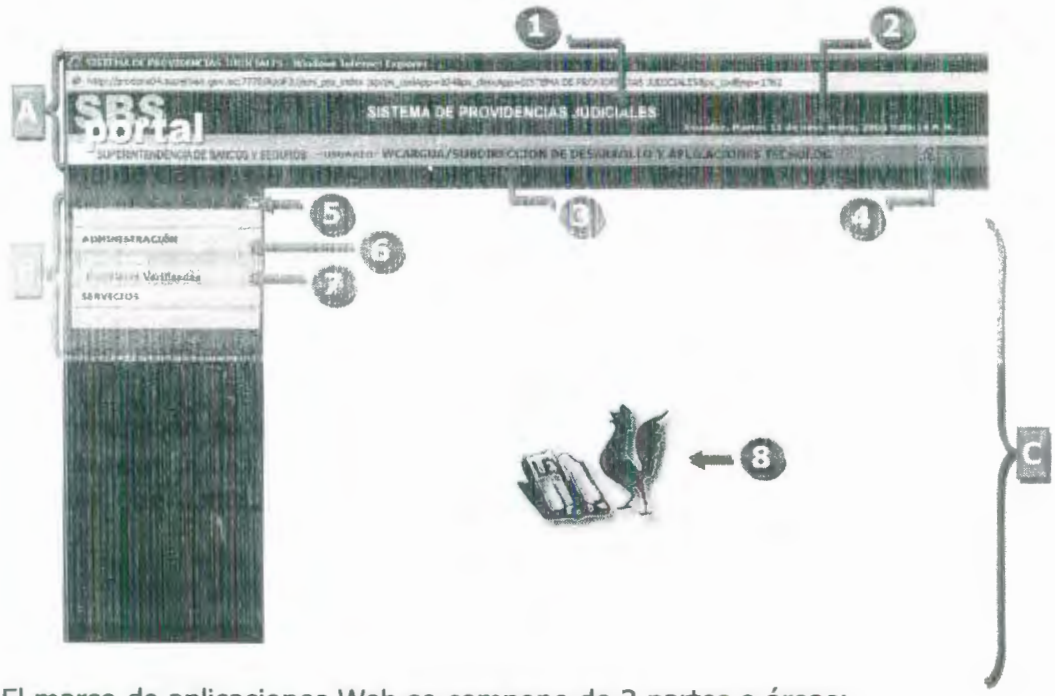
"ANEXO-6-Estándares visuales de páginas Web"

A6.1. Marco de aplicaciones Web

En la Superintendencia de Bancos y Seguros (SBS), en un proyecto liderado por el autor de esta tesis, se desarrolló un pequeño producto para la incorporación centralizada mediante el ingreso de parámetros en un esquema de base de datos, de todas las aplicaciones tipo Web para los servicios que se prestan al público en general como por ejemplo las consultas de Central de Riesgos.

Este se convierte en un estándar visual para el marco de los menús de opciones, perfiles y roles de acceso a todas las aplicaciones Web.

A continuación se presenta un ejemplo del marco de la aplicación Web de Providencias Judiciales, para explicar de forma sencilla todos los elementos visuales que componen el marco visual estándar de las aplicaciones Web de la SBS:



El marco de aplicaciones Web se compone de 3 partes o áreas:

A: Contiene información general de la aplicación Web, es decir, el logotipo de los servicios del Portal de la SBS, la descripción de la

aplicación, la fecha y hora, el nombre corto del usuario conectado a la aplicación, y un botón de manejo de funciones de la barra de menú.

La sección "A" contiene las siguientes partes:

1. Nombre de la aplicación Web.
2. Fecha y Hora.
3. Nombre corto del Usuario que se encuentra conectado a la aplicación.
4. Ícono de salida o cierre de la aplicación.

B: Esta sección contiene el menú de opciones correspondiente a un perfil o rol específico, definidos para cada aplicación Web.

La sección "B" contiene las siguientes partes:

1. Ícono para ocultar el menú de opciones de la aplicación.
2. Ejemplo de menú de opciones.
3. Opciones pertenecientes a un menú específico.

C: Esta sección corresponde al espacio o área de trabajo de la aplicación Web. En esta parte de la pantalla se descargaran y se visualizarán todos los componentes de las pantallas y reportes de la aplicación Web.

La sección "C" contiene las siguientes partes:

1. Al inicio de la aplicación Web se presenta en el centro de esta sección el logotipo de la Superintendencia de Bancos y Seguros.

A6.2. Estándares para el desarrollo visual de aplicaciones Web

Se debe conservar un esquema para el diseño en la web, para la página principal basado en el marco de aplicaciones Web.

Para los avisos de errores y mensajes de información se los presentará junto con imágenes que representen cada tipo de mensaje. Las imágenes que se deben presentar son:

Utilizado para mensajes que indican una acción exitosa en la aplicación.

Utilizado para mensajes que indican una acción de error en la aplicación.

Utilizado para mensajes que indican una acción de cancelación en la aplicación.

Utilizado para mensajes que indican una advertencia antes de ejecutar una acción en la aplicación.

Para los botones de consulta se utilizarán las siguientes imágenes:

Sirve para indicar que la pantalla ingresa a modo de consulta.

Sirve para ejecutar la consulta una vez que se han ingresado los valores de los criterios de búsqueda.

Utilizado para el botón de navegación que salta al primer registro de la consulta por pantalla.

Utilizado para el botón de navegación que salta al grupo de registros anteriores de la consulta por pantalla.

Utilizado para el botón de navegación que salta al registro anterior de la consulta por pantalla.

Utilizado para el botón de navegación que salta al registro siguiente de la consulta por pantalla.

Utilizado para el botón de navegación que salta al grupo de registros siguientes de la consulta por pantalla.

Utilizado para el botón de navegación que salta al último registro de la consulta por pantalla.

Para los botones transaccionales se utilizarán las siguientes imágenes:

Sirve para indicar que se realizará el ingreso de un nuevo registro.

Indica que la información de un registro será modificada.

Indica que la información de un registro será eliminada o borrada.

Indica una acción de cancelación y retorno a la pantalla inmediatamente anterior a la actual.

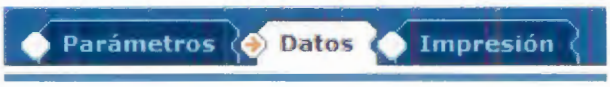
Sirve para indicar la acción de confirmación de almacenamiento de la información de la pantalla actual.

Para los botones relacionados con la ejecución e impresión de reportes se utilizarán las siguientes imágenes:

Sirve para indicar la ejecución de los reportes.

Indica la acción de impresión de los reportes.

Para la navegación de reportes con parámetros comunes se utilizarán pestañas de navegación, tal como se muestra en la siguiente imagen:



La apariencia y colores de los componentes visuales de las pantallas de las aplicaciones Web se basarán en las siguientes sentencias CSS:

Links

```
A:link { color: blue }        /* link sin visitar*/
A:visited { color: red }      /* links visitado*/
A:active { color: lime }      /* links activos*/
```

Texto de página

Texto normal de la página, tal como texto que indique el campo a ingresar.

LetraNormal { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 9px; font-style: normal; line-height: normal; font-weight: normal; font-variant: normal}

Texto de aviso

Texto que se va ha utilizar para presentar avisos de tipo informativo.

LetraAviso { color: Green; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-style: italic ; line-height: normal}

Texto de Errores

Texto que se va ha utilizar para presentar información de errores.

LetraError { color: Red; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-style: italic ; line-height: normal}

Texto de Título

Se utilizará éste formato para el texto que indique la opción de la página que ha ingresado el usuario, títulos.

LetraTitulo { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 16px; font-style: normal; line-height: normal; font-weight: bolder; font-variant: normal}

Texto blanco

Se utilizará éste formato para las consultas, se le asignará a la cabecera de la información a presentar.

LetraConsulta { color: White; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px; font-style: normal; line-height: normal; font-weight: bolder; font-variant: normal}

Texto de Construcción

Se utilizará éste formato para la información a presentar en una página que se encuentre en construcción.

LetraConstruccion { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 24pt; font-style: normal; line-height: normal }

Colores de tablas y bordes

Color de la cabecera que contiene la información de los datos a presentar en una consulta.

Consulta { background: #0055DD}

Color del borde de la tabla donde se presenta la información de consulta, esto se aplica para las sub-consultas de la página.

BoxRojo { border: #FF0000; border-style: solid; border-top-width: thin; border-right-width: thin; border-bottom-width: thin; border-left-width: thin}

Color del borde de la tabla, agrupando la información de un formulario de acuerdo su grado de complejidad y su categoría.

BoxNegro { border: #000000; border-style: solid; border-top-width: thin; border-right-width: thin; border-bottom-width: thin; border-left-width: thin}

Color del borde de la tabla para los avisos de error.

BoxAzul { border: thin #0055DD solid}

A6.2.1. Consideraciones especiales

En el caso de presentar un formulario de ingreso las partes de cabecera de consulta y presentación de información se unificarán.

En los Objetos input tipo text se deberá establecer la cantidad de dígitos que se podrán ingresar, en base a la longitud definida en la Tabla correspondiente de la Base de Datos.

El tamaño de los Objetos input tipo text será ajustado a la cantidad de dígitos permitidos.

Se deberán utilizar librerías comunes de validación en lenguaje JavaScript, para verificar el ingreso de acuerdo a la lógica de negocio definida para cada aplicación Web.

"ANEXO-7-Estándares de nomenclaturas y abreviaturas"

En este anexo se detallan los estándares propuestos para los nombres y abreviaturas de los directorios, archivos, estructura de directorios.

A7.1. Aspectos generales

El nombre de la aplicación está compuesto de dos partes: una abreviatura de tres letras mayúsculas y la descripción. Ejemplo:

PJU SISTEMA DE PROVIDENCIAS JUDICIALES

A7.1.1. Reglas de ortografía y de puntuación

- Los títulos no llevarán punto final.
- Las letras mayúsculas llevarán tildes según reglas ortográficas.
- Las tablas y cuadros deberán tener pies de ilustraciones sin punto final.
- En cuanto a las cifras, emplearemos la coma para separación de millares y el punto para separar los enteros con decimales

A7.1.2. Reglas tipográficas y de formato

- Se utilizará tipo de letra Tahoma de 12 puntos para títulos, subtítulos de 11 puntos, para el resto de texto de 10 puntos.
- Tahoma de 10 puntos con negrita si son necesarios otros títulos.
- Cada nuevo párrafo de un texto no debe llevar sangría inicial.
- Todo documento contendrá un encabezado y pie de página.
- En cada documento que se genere se deberá especificar fecha de inicio, fecha de entrega, personas responsables.
- En cuanto al uso de las comillas, recurriremos al uso de comillas dobles.
- Todas las figuras y tablas deben estar numeradas y tener título, y cada uno de ellas estará referenciada en el texto, al menos una vez.

- Para numeraciones usaremos números, letras, y guiones.

A7.2. Nombres de estructuras de directorios y directorios

Para almacenar los archivos de un nuevo desarrollo de una aplicación Web, se deberá crear un directorio en la carpeta raíz de la PC, cuyo nombre deberá coincidir con las siglas definidas para la aplicación. Ejemplo:

C:\PJU Carpeta de documentos del Sistema de Providencias Judiciales

Luego se deben definir 3 directorios:

1. DOCUMENTACION
2. FUENTES-JAVA
3. FUENTES-REPORTES

1. **DOCUMENTACION.-** Se deben definir 4 carpetas, es decir, un directorio por cada fase de la metodología, para almacenar por cada fase su correspondiente documentación.

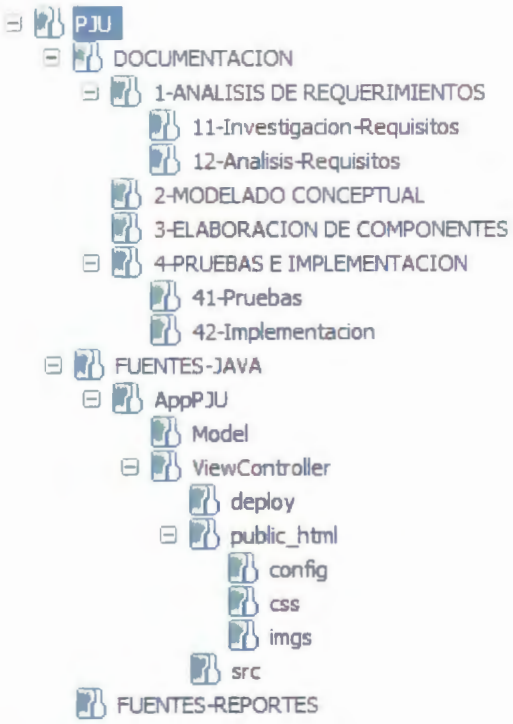
   Para la fase de Análisis de Requerimientos se debe definir 2 subdirectorios por cada sub-etapa, de forma similar se lo debe hacer por la fase de Pruebas e Implementación.

2. **FUENTES-JAVA.-** Se debe definir una carpeta contenedora de los fuentes autogenerados por la herramienta IDE de Desarrollo de la aplicación Web. Ejemplo:

   AppPJU

3. **FUENTES-REPORTES:** Dentro de este directorio se deberán almacenar los archivos correspondientes a los reportes de la aplicación.

A continuación se presenta un ejemplo visual de un ejemplo de estructura de directorios para la aplicación Web de Providencias Judiciales:



A7.3. Nombres y abreviaturas propuestos para cada fase de la metodología

A7.3.1. Archivos físicos

Para una identificación de los archivos que se generen se utilizará el siguiente formato:

AAA_BCCC_##.xxx

El detalle del formato se encuentra a continuación:

Nombre	# Caracteres	Descripción	Ejemplo
AAA	3	Nombre del proyecto o aplicación	PJU WGP
B	1	Fase del proyecto en la cual se emite el documento	A Análisis de requerimientos M Modelado Conceptual C Elaboración de Componentes P Pruebas e Implementación

CCC	3	Descripción del documento	DIC - Diseño del Sistema (Clases) DIE - Diagrama de Estados DII - Diagrama Interacción DPU - Documento de Pruebas FAC - Documento de Factibilidad FOR - Formularios MTE - Manual Técnico MTI - Manual de instalación y Despliegue MUS - Manual de Usuario PLT - Plantillas SOP – SOPORTE
##	2	Registro de versión del documento	01
.xxx	3	Extensión del archivo dependerá del software que se utilice	doc xls mpp

A7.3.2. Identificación de diagramas

Los diagramas deberán sujetarse al siguiente esquema:

AANnn...

AA: Categoría del diagrama
Nnn: Nombre descriptivo de 30 caracteres máximo.

Tenemos a continuación las diferentes categorías de diagramas:

Categoría	Diagrama
CU	Casos de Uso
CL	Clases
IT	Interacción
ET	Estados

Ejemplo:

CUDescargaProvidencia: Caso de Uso del proceso de descarga del Documento de la Providencia Judicial.

"ANEXO-8-Estándares de instalación y despliegue"

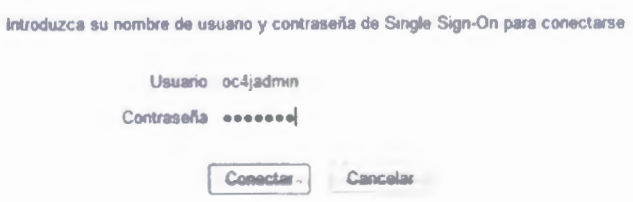
En este anexo se incluye las guías para realizar el despliegue e instalación de las aplicaciones Web en el ambiente de Producción y Desarrollo de aplicaciones.

Se ha especificado las instrucciones estándares de un manual técnico de instalación en consenso con los técnicos del área de administración de recursos tecnológicos.

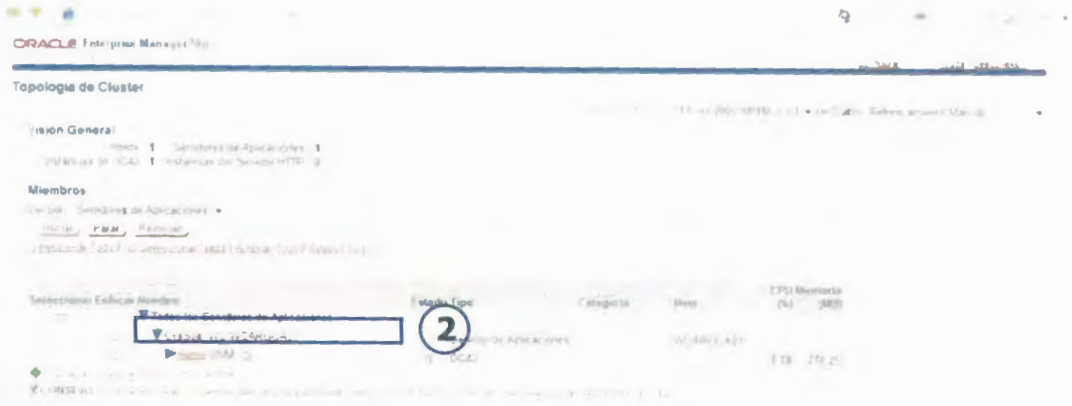
A continuación se presenta un ejemplo práctico de la elaboración de un manual técnico para la instalación de un software específico y que sirvió como ejemplo para la elaboración de esta tesis.

Manual de Instalación de la aplicación de Providencias Judiciales (PJU) (versión Web 1.0)

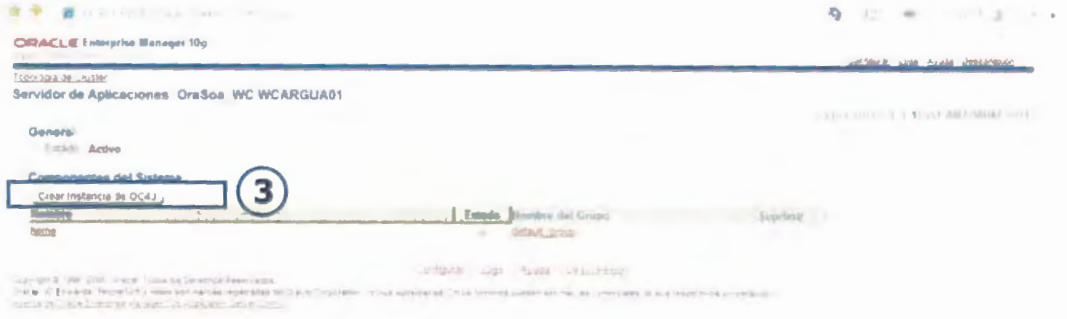
1. Ingresar al Application Server Control del **SOA** con el usuario **oc4jadmin.**



2. En la pantalla de **Topología de Cluster** en la sección **Miembros**, dar clic en el enlace de la instancia del Servidor de aplicaciones.



3. Luego se mostrará la pantalla de instancias OC4Js instaladas en la instancia del Servidor de Aplicaciones. Dar clic en el botón **Crear instancia de OC4J**.



4. En la pantalla de creación de la nueva instancia OC4J ingresar los siguientes parámetros:

4.1. Ingresar el nombre de la instancia:

Nombre de la instancia de OC4J OC4J_SERV_APP

4.2. Dar clic en la opción **Agregar a Nuevo Grupo con Nombre** y luego ingresar el nombre del grupo:

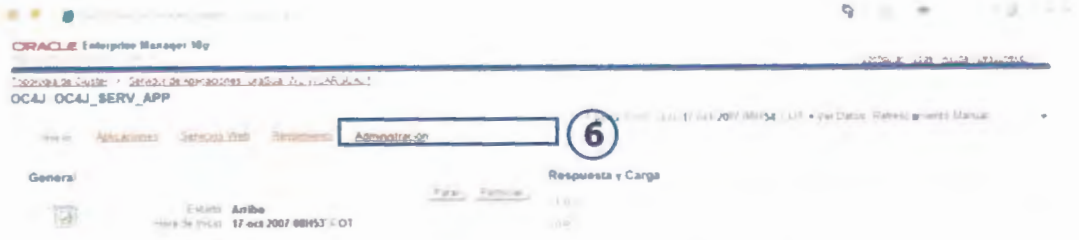
Nuevo Nombre de Grupo GRP_SBS_INTERNO

4.3. Marcar el checkbox para que se Inicie la instancia OC4J luego de la creación.

4.4. Dar clic el botón **Crear**.

5. En la siguiente pantalla se muestra un mensaje de confirmación de la creación de la nueva instancia OC4J. Dar clic sobre el enlace de la nueva OC4J.

6. Dar clic sobre la pestaña de **Administración**.



7. En la pantalla de Administración del OC4J dar clic en el botón correspondiente al vínculo **Propiedades del Servidor**.

8. En la pantalla de **Propiedades del Servidor** de la nueva instancia OC4J cambiar los siguientes parámetros.

8.1. En el valor del Puerto se debe ingresar el número **7778**.

8.2. Para el protocolo se debe seleccionar de la lista de valores disponible el protocolo **http.**

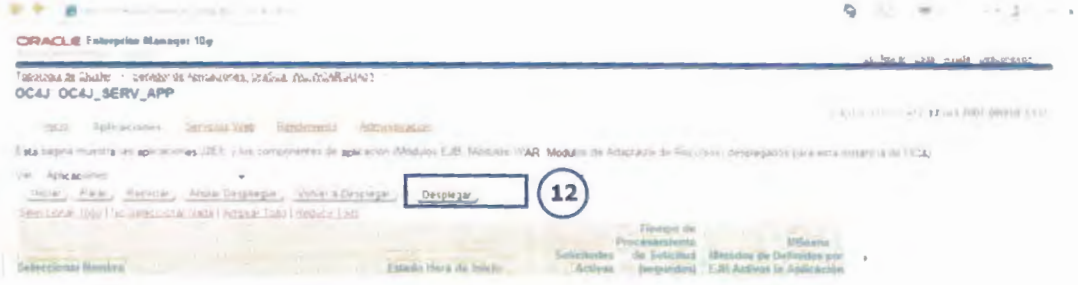
8.3. Luego dar un clic sobre el botón **Aplicar**.

9. Luego en la misma pantalla de Administración se muestra un mensaje de confirmación de los cambios realizados en las propiedades del Servidor. Dar clic en el enlace **Topología de Cluster**.

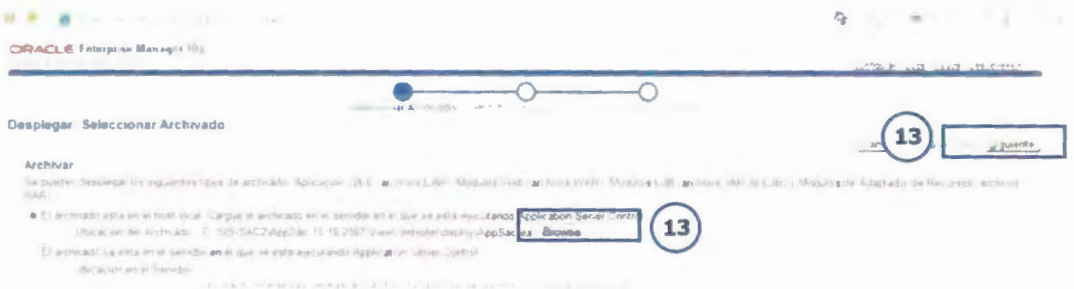
10. En la pantalla de **Topología de Cluster** marcar el checkbox correspondiente a la nueva instancia OC4J creada y luego dar clic en el botón **Iniciar**.

11. Se muestra el mensaje de confirmación de inicio de los miembros de la topología. Luego dar clic en el **enlace de la nueva instancia OC4J**.

12. Dar clic en la pestaña de **Aplicaciones** de la pantalla de la nueva instancia OC4J, y luego dar clic en el botón **Desplegar**.



13. Seleccionar el archivo AppPJU.ear ubicado en **\\webpre\IAS\Produccion\OC4J_SERV_APP** dando clic en el botón **Browse** o **Examinar**, y luego dar clic en el Botón **Siguiente**.



14. En la pantalla de Atributos de Aplicación ingresar los siguientes parámetros:

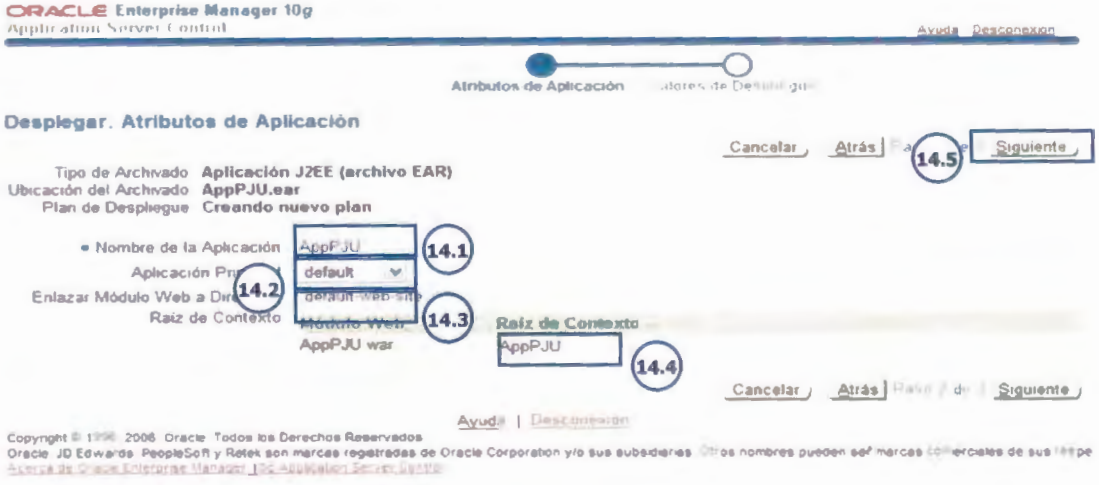
14.1. En el nombre de la aplicación ingresar **AppPJU**.

14.2. En el atributo **Aplicación Principal** seleccionar en la lista de valores **default**.

14.3. Para el atributo **Enlazar Módulo Web a Dirección** seleccionar en la lista de valores **default-web-site**.

14.4. En el atributo **Raíz de Contexto** ingresar el nombre **AppPJU.**

14.5. Clic en el botón **Siguiente**.



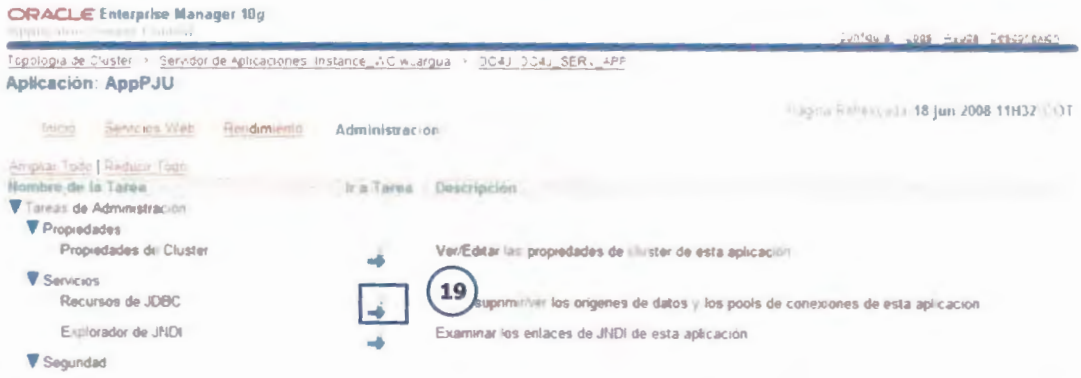
15. En la pantalla Valores de Despliegue aceptar los valores por default dando clic en el botón **Desplegar**.

16. Se muestra una pantalla con el log del despliegue, luego de que se muestra el mensaje de confirmación de que se ha desplegado correctamente dar clic en el botón **Volver**.

17. Después de dar clic en el botón **Volver** de la pantalla anterior, se muestra la pantalla de **Aplicaciones** pertenecientes al nuevo OC4J, entonces dar un clic en el **enlace AppPJU**.

18. En la pantalla de la **Aplicación: AppPJU** dar clic en la pestaña **Administración**.

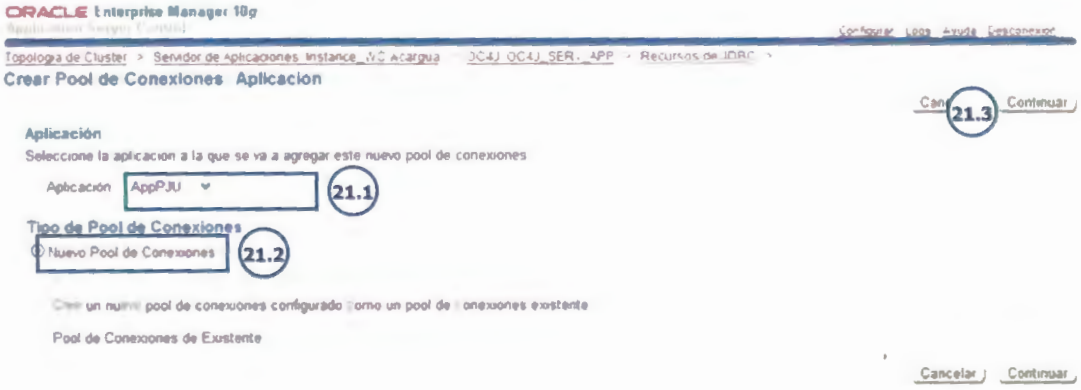
19. En la pantalla de **Administración** de la Aplicación:AppPJU dar clic en el botón correspondiente al enlace **Servicios/Recursos de JDBC**.



20. En la pantalla de **Recursos de JDBC** dar clic en el botón **Crear** de la sección **Pools de Conexiones**.

21. Escoger las siguientes opciones en la pantalla de creación de pool de conexiones:

21.1. En la lista de valores de **Aplicación** seleccionar **AppPJU**.
21.2. Para el **Tipo de Pool de Conexiones** escoger **Nuevo Pool de Conexiones**.
21.3. Dar clic en el botón **Continuar**.



22. Ingresar los siguientes parámetros para la creación del Pool de Conexiones:

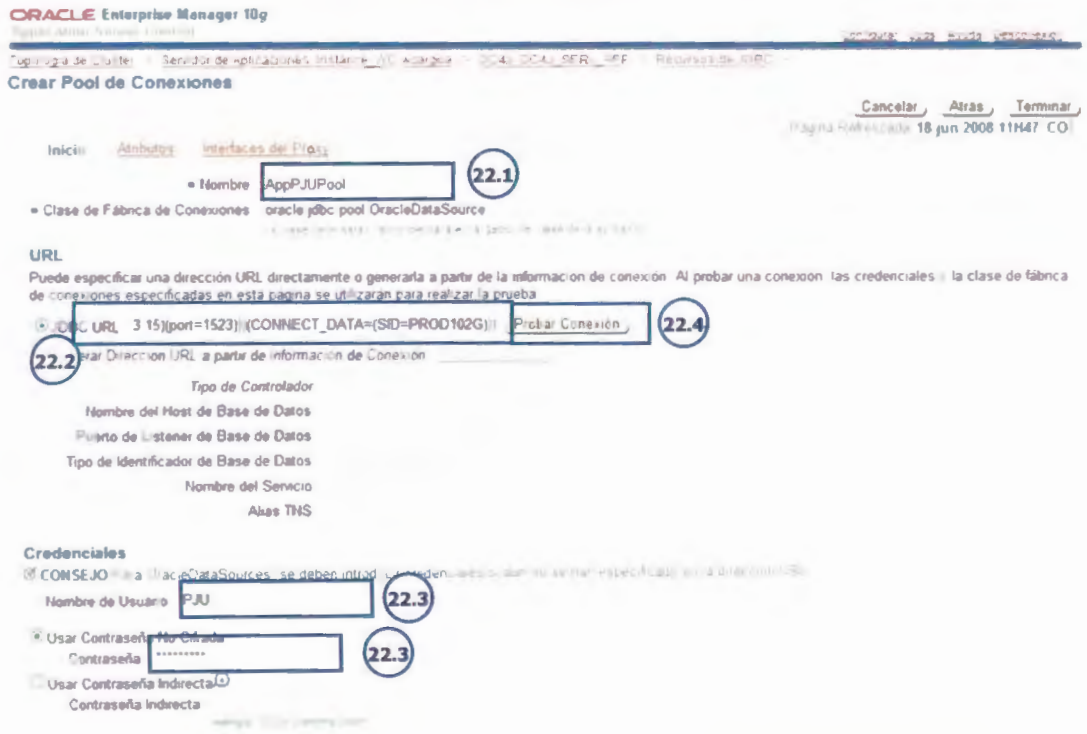
22.1. Para el **Nombre** ingresar **AppPJUPool**.

22.2. Sobrescribir el atributo **JDBC URL** con el valor:

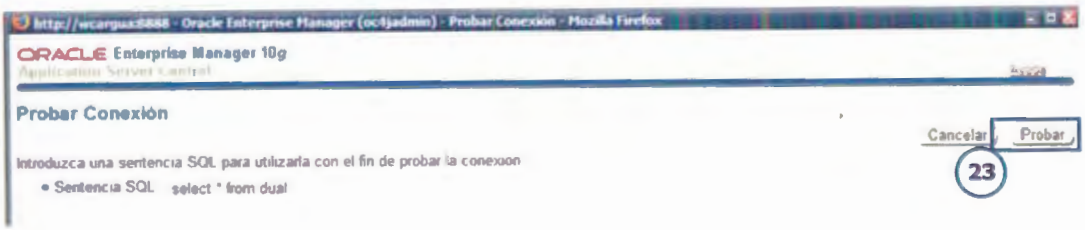
jdbc:oracle:thin:@(DESCRIPTION=
(ADDRESS_LIST=(LOAD_BALANCE=on)(FAILOVER=on)(ADDRESS=(PROTOCOL=TCP)(HOST=prodora01-vip.superban.gov.ec)(port=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=prodora02-vip.superban.gov.ec)(port=1521)))(CONNECT_DATA=(SERVICE_NAME=prod102g.superban.gov.ec)))

22.3. Luego ingresar el Nombre de Usuario: **PJU** y en la Contraseña No Cifrada **la clave del Usuario PJU PRODUCCION.**

22.4. Dar clic en el botón **Probar Conexión**.



23. Dar clic en el botón **Probar**.



24. Debe salir un mensaje de confirmación indicando que la conexión se estableció correctamente, luego dar clic en el botón **Terminar**.

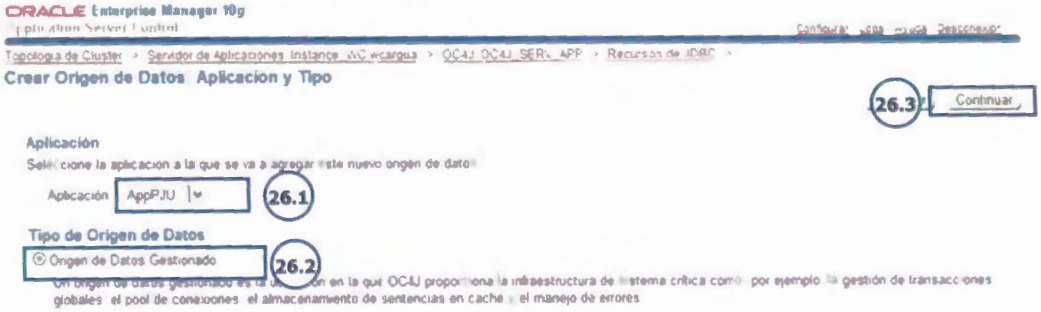
25. Dar clic en el botón **Crear** en la Sección **Orígenes de Datos**.

26. Seleccionar los siguientes valores para crear el origen de datos:

26.1. Para el atributo **Aplicación** seleccionar **AppPJU**.

26.2. En el Atributo **Tipo de Origen de Datos** seleccionar **Origen de Datos de Gestionado**.

26.3. Dar clic en el botón **Continuar**.



27. Ingresar los siguientes valores para continuar la creación del origen de datos:

27.1. En el nombre del Origen de datos poner: **AppPJU**.

27.2. Para el atributo **Ubicación de JNDI** ingresar: **jdbc/AppPJUDS**
27.3. En la lista de valores de **Pool de Conexiones** seleccionar **AppPJUPool**.
27.4. Dar clic en el botón **Terminar**.

28. Dar clic en el enlace Servidor de Aplicaciones.

29. Dar un clic en el enlace de la instancia OC4J creada recientemente.

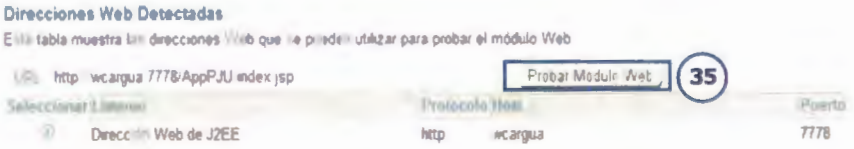
30. Clic en la pestaña de **Aplicaciones**.

31. Clic en la Aplicación **AppPJU**.

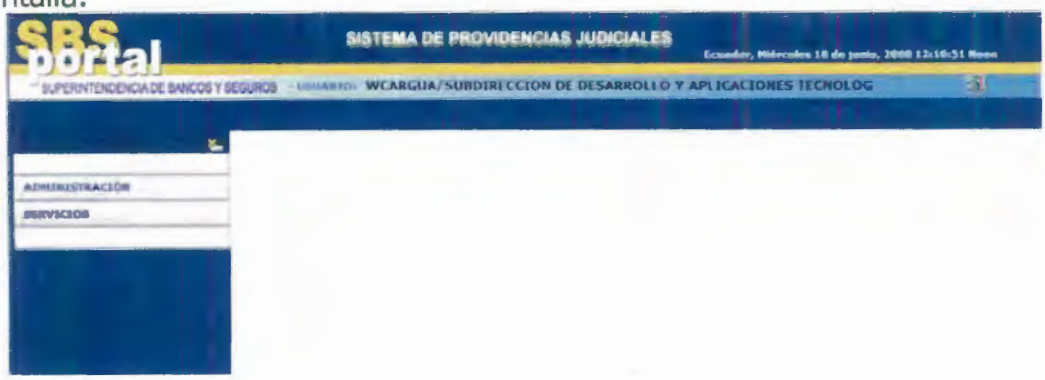
32. Dar un clic en el enlace del **módulo** Web **AppPJU**.

33. Clic en el botón **Probar Módulo Web**.

34. Luego de agregar el nombre de la página de inicio de la aplicación al URL, dar un clic en el botón **Probar Módulo Web**.



36. El resultado de la prueba del módulo Web debe ser similar a la siguiente pantalla:



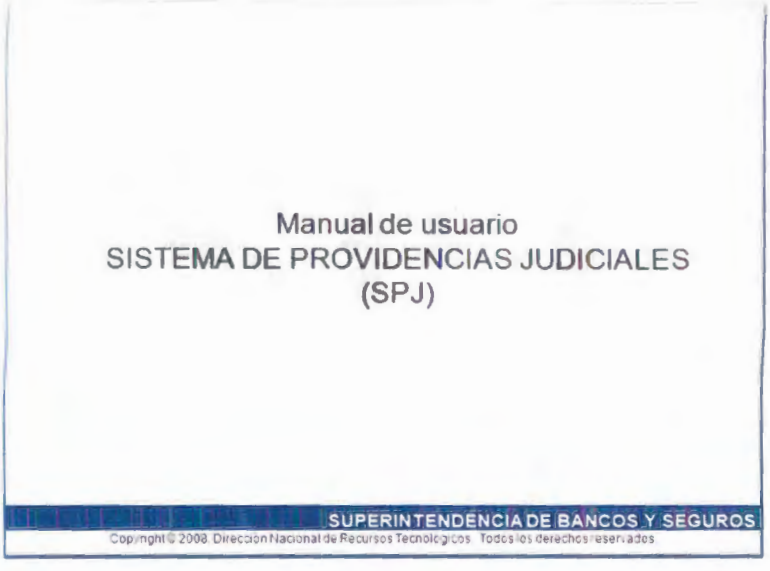
Fecha: *viernes, 18 de julio de 2008*
Elaborado por: *LSI. William Cargua Freire*

"ANEXO-9-Guía para realizar el manual de usuario de las aplicaciones"

En este anexo se incluye las guías para realizar el mapa de usuario de las aplicaciones Web.

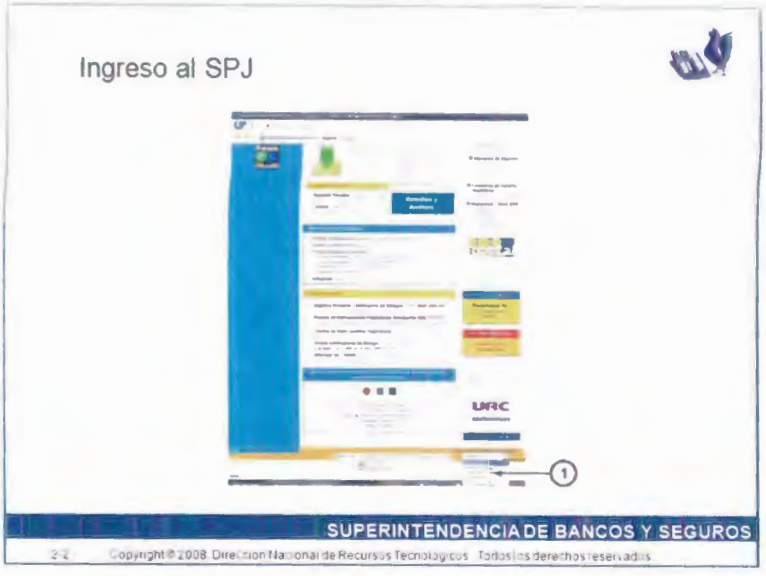
Se ha especificado las instrucciones estándares para redactar un manual de usuario de aplicaciones Web, en consenso con los técnicos del área de desarrollo y aplicaciones tecnológicas.

A continuación se presenta un ejemplo práctico de la elaboración de un manual de usuario para un software específico y que sirvió como ejemplo para la elaboración de esta tesis.



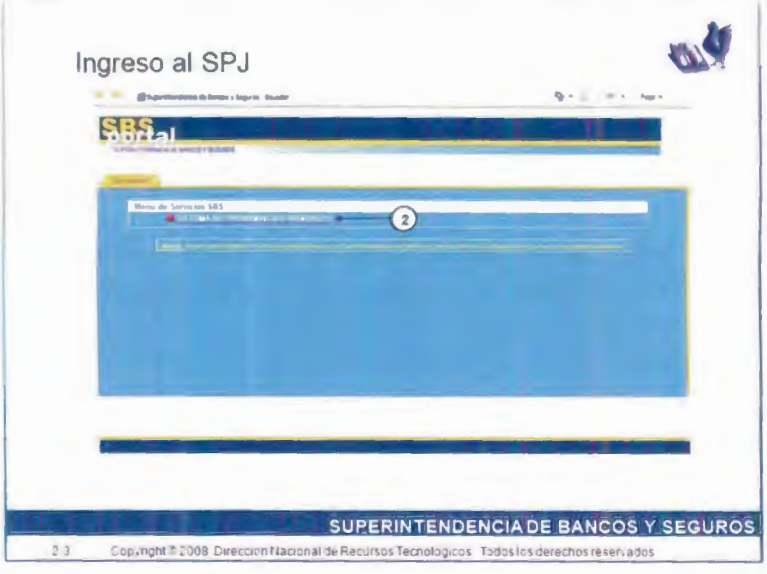
Este manual describe la manera como las instituciones supervisadas por la Superintendencia de Bancos y Seguros deben acceder al Sistema de Providencias Judiciales (SPJ).

Antes de acceder a este sistema, las instituciones supervisadas deben tener asignados los respectivos usuarios de acceso a la aplicación. Los usuarios de acceso deben ser emitidos por la Superintendencia de Bancos y Seguros, luego de completar el proceso de un nuevo usuario para acceso al mencionado sistema.



Para ingresar al sistema de Providencias Judiciales se debe visitar el sitio Web de la Superintendencia de Bancos y Seguros: www.superban.gov.ec.

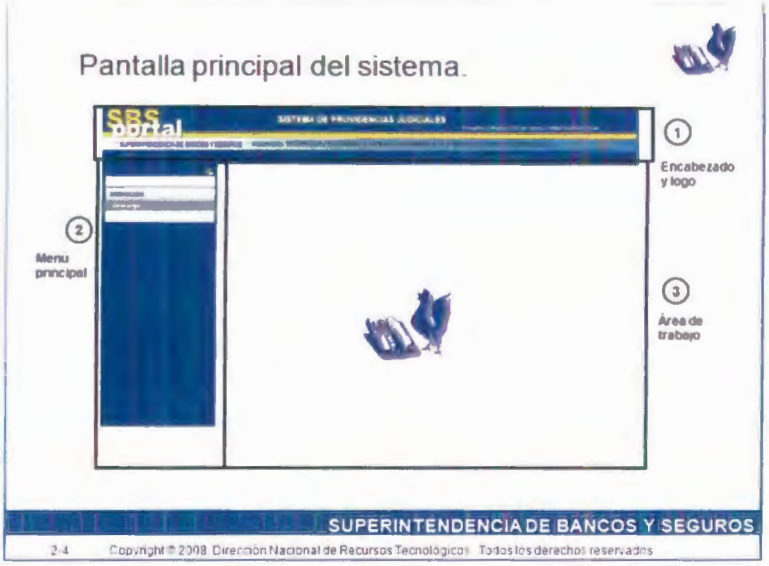
1. Entrada al Portal de servicios externos.- Luego de ingresar a la página Web, se debe hacer referencia a la sección de información a entidades controladas para ingresar a la pantalla del Portal de servicios de la Superintendencia de Bancos y Seguros.



Ya que el SPJ es una aplicación integrada al Portal de servicios externos de la Superintendencia de Bancos y Seguros, para acceder al mismo se debe

ingresar primeramente al Portal con su clave única de identificación, entregada a cada usuario de las instituciones del sistema supervisado.

Una vez dentro del portal, se debe navegar a la pestaña de "Servicios". En esta pantalla, se presentan links (ENLACES) a todos los sistemas a los que tenga acceso el usuario (ver número 2 en el gráfico). Al hacer clic en este vínculo, se abrirá una ventana con el sistema que se haya seleccionado.



La pantalla principal del POA está organizada de la siguiente forma:

1. Encabezado y logo.- Esta sección contiene el logotipo del Portal, además de información como la fecha actual, el usuario que está conectado al sistema.

2. Menú principal.- En esta sección se muestran las diferentes opciones del sistema, apareciendo éstas de acuerdo al "perfil" con el que ingresó el usuario.

3. Área de trabajo.- En esta área es donde se presentan las pantallas del sistema por cada una de sus opciones, con las que se realiza el trabajo o acciones respectivas.

El área de encabezado y logo de la pantalla principal del Sistema de Providencias Judiciales, está conformada de la siguiente manera:

1. Nombre del sistema.- Muestra el nombre de la aplicación.

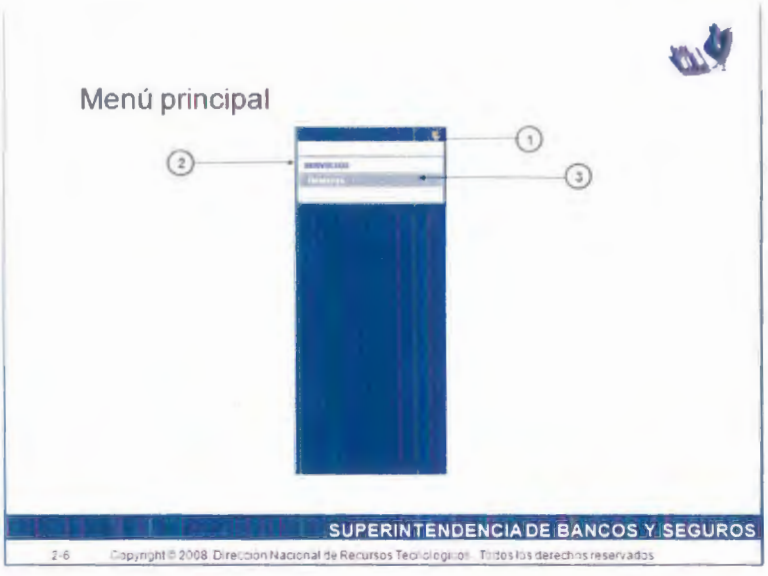
2. Logotipo del portal.- Se muestra el logotipo del Portal Interno que indica que la aplicación está integrada a éste.

3. Usuario.- Este campo indica el usuario que inició la sesión del sistema, así como el área o departamento a la que pertenece.

4. Fecha actual.- Muestra la fecha actual mientras se está ejecutando el sistema.

5. Cerrar sesión.- Icono que permite cerrar la aplicación.

6. Restablecer menú.- Este icono se presenta cuando el área de menú principal está oculta. Haciendo clic en este icono, se volverá a restablecer la vista del menú principal en la pantalla principal.

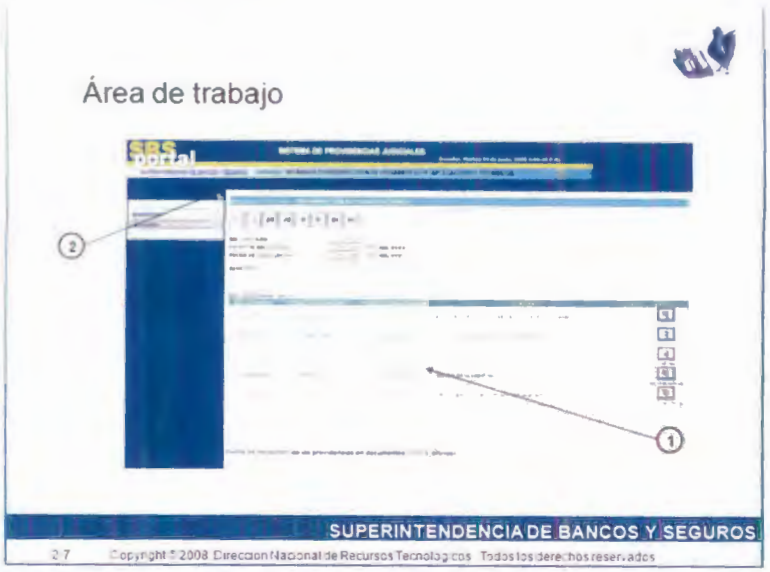


El área de menú principal del Sistema de Providencias Judiciales, está conformada de la siguiente manera:

1. Ocultar menú.- Haciendo clic en este icono, el menú principal se oculta, dejando más espacio disponible para el área de trabajo.

2. Submenú de opciones.- El menú principal está dividido a su vez por submenús que contienen las opciones del sistema, con el fin de una mejor organización.

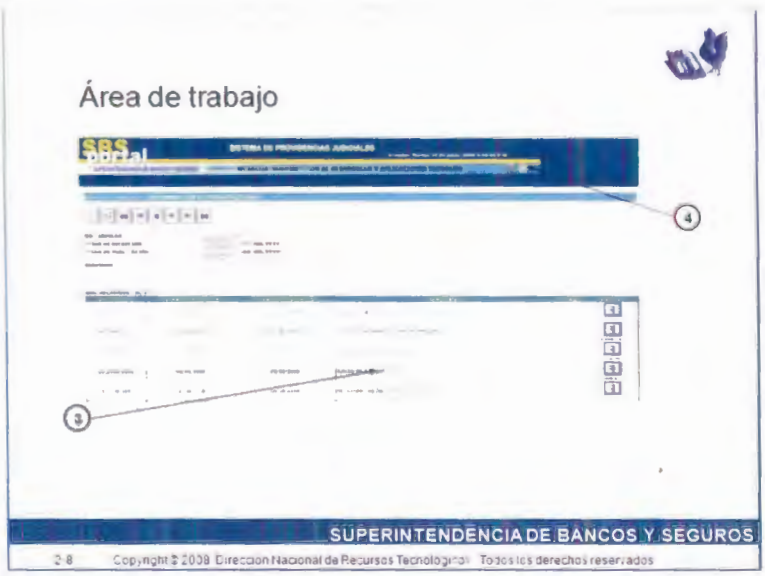
3. Opciones.- Finalmente tenemos las opciones del sistema, las mismas que son links (enlaces) a las diferentes pantallas del Sistema de Providencias Judiciales. Al hacer clic en uno de estos links, se cargará la pantalla correspondiente en el área de trabajo.



El área de trabajo es donde se muestran las pantallas con las que se desea trabajar, luego de haber hecho clic en los links (enlaces) de opciones como se describió en la lámina anterior. A continuación, una explicación de la pantalla mostrada:

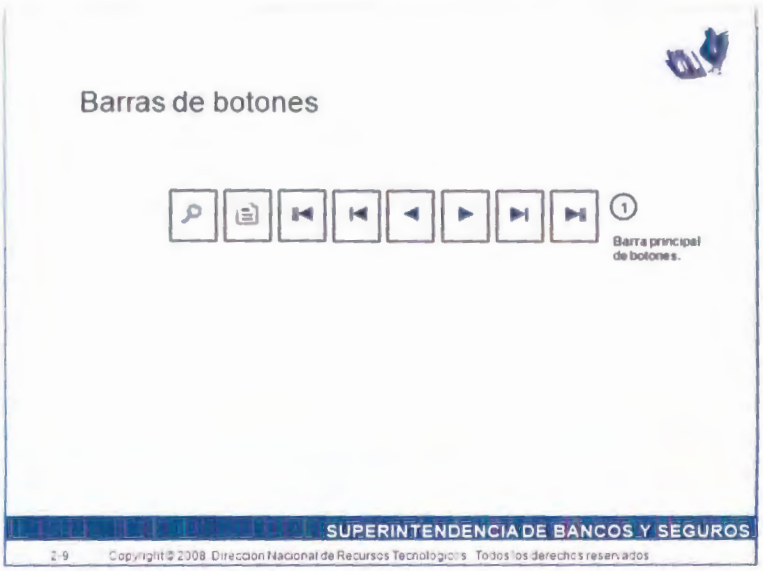
1. Pantalla cargada.- Se muestra como la pantalla seleccionada que se cargó o mostró en el área de trabajo.

2. Ocultar menú.- Si se desea hacer más amplia el área de trabajo, se puede hacer clic en este icono de manera que el menú principal se oculte y quede todo el ancho de la pantalla para la opción cargada o mostrada.



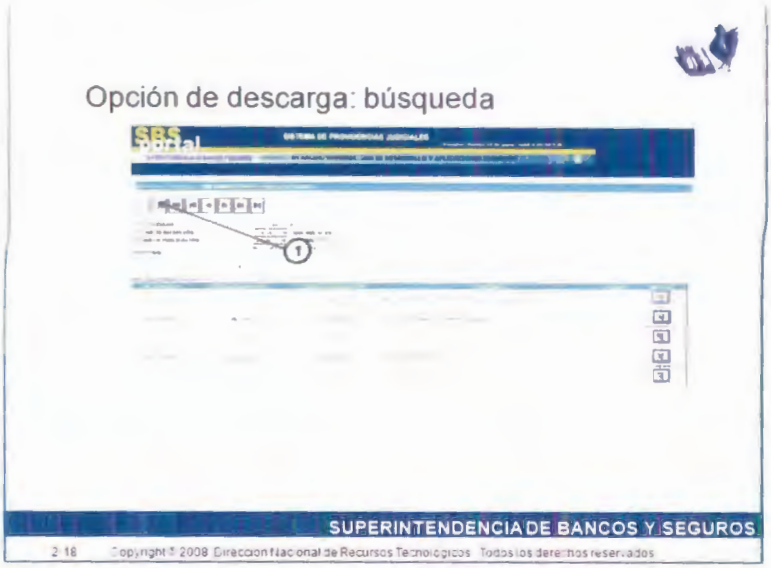
3. Menú oculto.- Se muestra en el espacio señalado como el área de trabajo ocupa el ancho de la pantalla una vez que se ha ocultado el menú principal del sistema.

4. Restablecer menú.- Si se desea volver a mostrar el menú, se debe hacer clic sobre este icono, de manera que la pantalla muestre el menú principal del sistema.



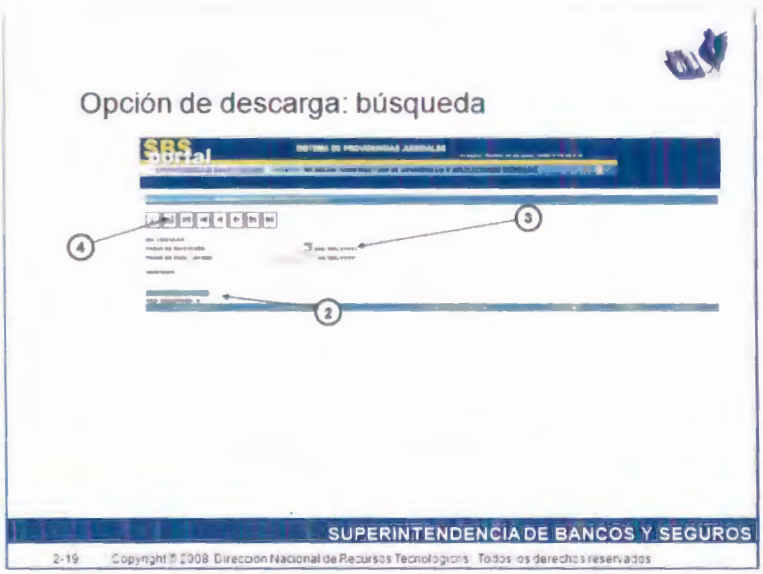
Dentro del SPJ podemos encontrar dos tipos de barras de botones que son las más comunes, esta son:

1. Barra principal de botones.- Esta aparece en la pantalla principal de cada opción (la primera que se carga al hacer clic en el link de la opción). Contiene los botones nuevo, modo búsqueda, ejecutar consulta y los botones de navegación entre registros.



Para realizar una búsqueda en la pantalla de descargas, se deben seguir los siguientes pasos:

1. Hacer clic en botón de modo búsqueda.- el primer paso es poner la pantalla en modo de búsqueda, esto se lo logra haciendo clic en el botón indicado en el gráfico.

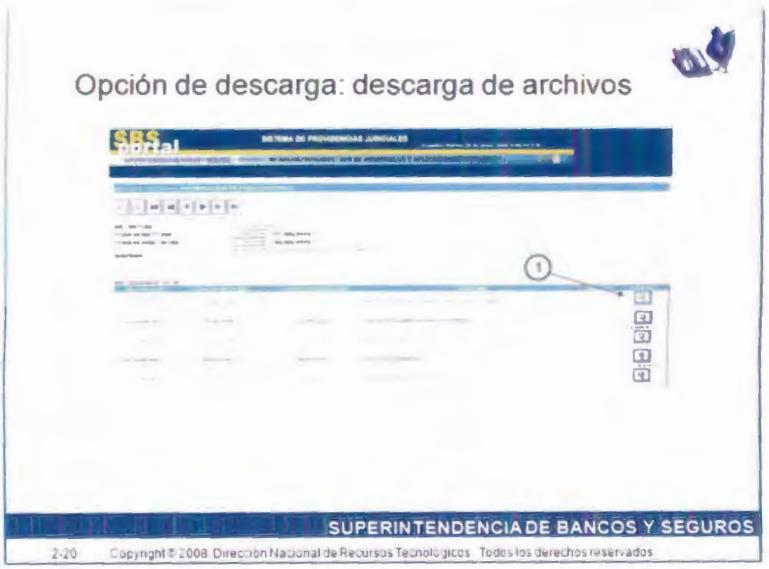


2. Modo de búsqueda.- Luego del paso uno, la pantalla debe ahora estar en modo búsqueda. Para confirmar esto, debe quedar en blanco el detalle de la pantalla como se muestra en el gráfico y por otro lado en la parte superior de la pantalla se activa los campos de criterios

para el ingreso por teclado, y el ícono de calendario para el ingreso de las fechas.

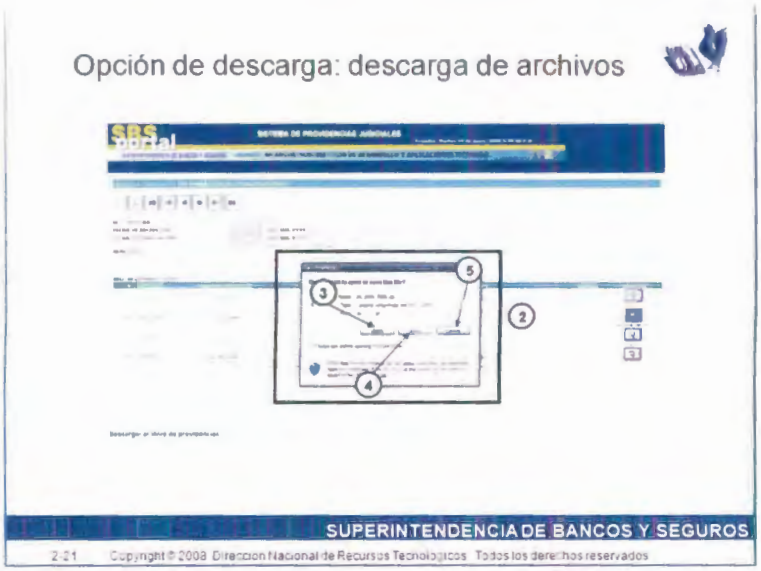
3. ***Ingresar criterio de consulta.-*** Estando la pantalla en modo búsqueda, procederemos a ingresar los criterios de consulta en el área correspondiente. Según la opción, se puede llenar uno o todos los campos de criterio de consulta que aparezcan.

4. **Ejecutar consulta.-** Una vez ingresados los criterios de búsqueda, debemos hacer clic en el botón "ejecutar consulta" para que se muestren los resultados de nuestra búsqueda en la pantalla principal de la opción.



Para realizar la descarga de la publicación electrónica indicada en un registro específico, se deben seguir los siguientes pasos:

1. **Hacer clic en icono de descarga de archivos.-** *El primer paso hacer clic en el icono mostrado en el gráfico.* Cabe anotar que no es necesario que el "indicador de registro actual" esté posicionado sobre el registro desde el cual queremos realizar la descarga.



Luego de dar clic sobre el ícono de descarga de archivos se presenta un cuadro de diálogo en el cual se detallan las siguientes opciones:

2. **Cuadro de diálogo de descarga.-** Esta sección se trata de un cuadro de diálogo con tres opciones para el usuario. Este cuadro de diálogo aparece inmediatamente luego de dar un clic sobre el ícono de descarga de los archivos.

3. **Abrir.-** Mediante este botón se puede realizar la apertura del archivo empaquetado.

4. **Guardar.-** Mediante este botón se puede guardar el archivo de la descarga en el sistema de archivos local de la computadora en donde se esté ejecutando el Sistema de Providencias Judiciales.

Fecha: miércoles, 23 de julio de 2008
Elaborado por: LSI. William Cargua Freire

"ANEXO-10-Guía para realizar el manual técnico de las aplicaciones"

En este anexo se incluye las guías para realizar el mapa técnico de las aplicaciones Web.

Se ha especificado las instrucciones estándares para redactar un manual técnico de aplicaciones Web, en consenso con los técnicos del área de desarrollo y aplicaciones tecnológicas.

A continuación se presenta un ejemplo práctico de la elaboración de un manual técnico para un software específico y que sirvió como ejemplo para la elaboración de esta tesis.

MANUAL TÉCNICO

SISTEMA DE PROVIDENCIAS JUDICIALES

Versión: 1.00

Elaborado por	LSI. William Cargua Freire	**Fecha:** 01 de agosto de 2008	
Actualizado por	--	**Fecha:** --	**Datos de cambios:** --

MANUAL TÉCNICO

IDENTIFICACIÓN DEL SISTEMA

Nombre del Sistema: SPJ, Sistema de Providencias Judiciales

Objetivo General: Simplificar el proceso de notificación, es decir proporcionar información confiable, oportuna y segura en la entrega de las providencias judiciales y ejercer un mejor control sobre el repudio de la recepción de las providencias por parte de las entidades del sistema supervisado por la SBS.

Esquemas que interactúan: SAC, HRA, SEO, GEN

CONTENIDO TÉCNICO

Versión herramientas de desarrollo:

Herramienta de desarrollo:	Jdeveloper 10.1.2

Base de datos:	Oracle 10g

Definición de paquetes importantes creados por el programador para el Application Server:

PAQUETE	DESCRIPCIÓN
AppPJU.model	Contiene las siguientes clases: Módulo de aplicaciones => AppModulePJU, ViewObjects => VWCONSULTAINSTITUCION, VWCONSULTAPAQUETE, VWEDITDATAPAQUETE, VWEDITDOCUMENTO, VWEDITPAQUETE, VWESTADOBUZON, VWPAQUETE, VWPAQUETEDOCUMENTO, VWPAQUETEINSTITUCION.
AppPJU.view	Contiene las siguientes clases: Componentes Action => Sbs_pju_inicioBuzonAction, Sbs_pju_modificaBuzonAction, Sbs_pju_nuevoBuzonAction, Sbs_pju_revisaBuzonAction, Sbs_pju_revisaEstAction.
sbs.pju.beans.com	Contiene las siguientes clases: JavaBeans => BAcceso, BConexion, Binfo, BprocesaED, BSession.
sbs.pju.beans.utils.com	Contiene las siguientes clases: JavaBeans => BDocumentos, BUtils, LobUtils, Secuencia.
sbs.pju.servlets.com	Contiene las siguientes clases: HttpServlets => SDocumentosMgr, SIngresoApp, SrevisaBuzon.

Definición de clases:

NOMBRE DE LA CLASE	DESCRIPCIÓN
AppPJU.model.VWCONSULTAINSTITUCION	Sirve para listar los buzones de descarga de providencias judiciales para todas las entidades del sistema supervisado por la SBS.
AppPJU.model.VWCONSULTAPAQUETE	Sirve para realizar la consulta interna de las descargas de providencias judiciales.
AppPJU.model.VWEDITDATAPAQUETE	Sirve para consultar los datos específicos de una providencia judicial.
AppPJU.model.VWEDITDOCUMENTO	Indica el número de fojas correspondiente a una providencia judicial.
AppPJU.model.VWEDITPAQUETE	Sirve para consultar los datos generales de una providencia judicial.
AppPJU.model.VWESTADOBUZON	Indica los estados de las descargas de las providencias judiciales.
AppPJU.model.VWPAQUETE	Sirve para controlar la existencia de providencias judiciales asignadas a cada entidad supervisada.
AppPJU.model.VWPAQUETEDOCUMENTO	Sirve para controlar la existencia del número de fojas para las providencias judiciales asignadas a cada entidad supervisada.
AppPJU.model.VWPAQUETEINSTITUCION	Sirve para consultar y asignar las publicaciones de providencias judiciales a cada entidad supervisada.

Definición de triggers y procedimientos o funciones de la base de datos:

ESQUEMA	NOMBRE OBJETO	DESCRIPCIÓN
PJU	TRpju_PjuPaquete.PJUPAQUETE	Sirve para guardar la auditoria de las acciones de INSERT, UPDATE, DELETE en la tabla PJUAUDPAQUETE
PJU	TRPJU_PJUPAQUETEDOCUMENTO.PJUPAQUETEDOCUMENTO	Sirve para guardar la auditoria de las acciones de INSERT, UPDATE, DELETE en la tabla PJUAUDPAQUETEDOCUMENTO
PJU	TRPJU_PJUPAQUETEINSTITUCION.PJUPAQUETEINSTITUCION	Sirve para guardar la auditoria de las acciones de INSERT, UPDATE, DELETE en la tabla PJUAUDPAQUETEINSTITUCION
PJU	PRC_PJUENVIAMAILNOTIFICACION	Ejecuta el envío de correos electrónicos por las acciones de modificación y eliminación de la información de una providencia judicial. Sólo se envían los títulos de las providencias en los correos electrónicos.
PJU	PRC_PJUMAILPROVIDENCIAS	Ejecuta el envío de correos electrónicos en dos JOBS que se ejecutan a las 12H00 y a las 16H00. Los envíos de mails se dan en dos paquetes acumulados, es decir, las publicaciones ingresadas hasta las 12H00 se envían en un correo electrónico desde el primer JOB y el acumulado de las publicaciones ingresadas en el día se envía en un mail en el JOB de las 16H00. Sólo se envían los títulos de las providencias en los correos electrónicos.

"ANEXO-11-Formulario de bitácora de reuniones del prototipo de software"

BITACORA DE REUNION

Bitácora de reuniones en el proceso de desarrollo de aplicaciones de la Superintendencia de Bancos y Seguros, Dirección Nacional de Recursos Tecnológicos, Subdirección de Desarrollo y Aplicaciones Tecnológicas.

Lugar:	Secretaría General
Fecha:	martes, 03 de junio de 2008
Horario:	09H00 – 13H00 y 14H00 – 16H00
Proyecto:	PROVIDENCIAS JUDICIALES
Etapa:	INVESTIGACIÓN DE REQUISITOS
Tema a tratar:	CARACTERÍSTICAS DEL PROCESO MANUAL DE NOTIFICACIÓN DE PROVIDENCIAS JUDICIALES
Asistentes:	LCDA. LETTY SUÁREZ CARRASCO, LCDO. PABLO COBO LUNA, LSI. WILLIAM CARGUA FREIRE

1. Asuntos tratados en la reunión

- **Agenda**

  Se realizaron preguntas acerca del funcionamiento del proceso de notificación de las providencias judiciales, hacia las entidades controladas por la SBS.

  Una vez que se hizo un reconocimiento de las funcionalidades de proceso de notificación de providencias judiciales hacia las entidades controladas, se realizó una pequeña lluvia de ideas con respecto a las posibles funcionalidades del proceso que podrían ser automatizadas.

- **Restricciones**

  Se encontraron una posible limitación, que consiste en que pudiera darse un cambio de normativa que pudiera eliminar el proceso de notificación de providencias judiciales. Este cambio podría sólo ser dispuesto por el momento por la Presidencia de la República.

2. Pendientes.

Para la próxima reunión se indicará la primera versión del prototipo, que incluye la funcionalidad completa del primer módulo de la aplicación de providencias judiciales.

3. Conclusiones

Se obtiene la autorización de los usuarios y del Director Nacional de Recursos Tecnológicos de la SBS, para que a partir del día siguiente se dé inicio a la elaboración del prototipo de la aplicación de notificación automatizada de Providencias Judiciales a las entidades controladas por la SBS.

4. Próxima reunión.

Se acordó mantener una próxima reunión para el día jueves 31 de julio de 2008.

5. Acuerdos alcanzados

Los usuarios estuvieron de acuerdo en la forma en cómo serían automatizadas ciertas funcionalidades en la nueva aplicación para la notificación automática de las providencias judiciales a las entidades del sistema controlado.

Los acuerdos alcanzados en esta reunión son la base para el desarrollo del proyecto.

Atentamente,

SDAT

LSI. William Cargua Freire
Desarrollo de aplicaciones

ÁREA USUARIA

LCDA. Letty Suárez Carrasco
Secretaría General

LCDO. Pablo Cobo Luna
Secretaría General

"ANEXO-12-Documento de Requisitos de Software del prototipo"

DOCUMENTO DE REQUISITOS DE SOFTWARE
FECHA DE ELABORACIÓN: 06/06/2008

1. Introducción

La Superintendencia de Bancos y Seguros mediante la unidad de Secretaria General mantiene un proceso manual para realizar la notificación de las providencias judiciales a las entidades controladas.

Para iniciar este proceso de notificación, secretaria general recepta las providencias judiciales debidamente legalizadas mediante oficios o con la descripción de su alcance en las localidades de la Superintendencia de Bancos y Seguros. Cabe señalar que esta recepción de documentos se la realiza con una frecuencia diaria.

Luego una persona encargada en Secretaria General realiza la redacción de una circular en donde se indica cuáles es o son las providencias judiciales recibidas y que luego serán objeto de revisión por parte de las entidades financieras controladas.

Una vez terminada la redacción de la circular una persona encargada de Secretaria General realiza la reproducción de copias fotostáticas de la o las providencias recibidas en un número aproximado de 220 copias según el número de providencias que se reciben diariamente. Después la persona encargada organiza la o las providencias fotocopiadas con la circular respectiva y las ajunta en un sobre para su posterior envío.

Los sobres con los documentos son enviados mediante una empresa externa de transporte y entrega, dirigidos a todas las entidades financieras controladas.

2. Participantes en el Proyecto

Las personas participantes del Proyecto son las siguientes:

LSI. William Cargua Freire	SDAT
LCDA. Letty Suárez Carrasco	Secretaría General
LCDO. Pablo Cobo Luna	Secretaría General

3. Descripción del Sistema Actual

No existe aplicación informática.

4. Objetivos del Sistema

OBJ-01	***Mejorar el proceso de notificación***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	LCDA. Letty Suárez Carrasco
Descripción	Simplificar el proceso de notificación, es decir proporcionar información confiable, oportuna y segura en la entrega de las providencias judiciales.
Sub-objetivos	--
Importancia	Vital
Urgencia	Inmediatamente
Estado	En construcción
Estabilidad	Alta
Comentarios	--

OBJ-02	***Controlar el repudio de la información***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna
Descripción	Ejercer un mejor control sobre el repudio de la recepción de las providencias por parte de las entidades financieras controladas.
Sub-objetivos	--
Importancia	Vital
Urgencia	Inmediatamente
Estado	En construcción
Estabilidad	Alta
Comentarios	--

5. Catálogo de requisitos del Sistema

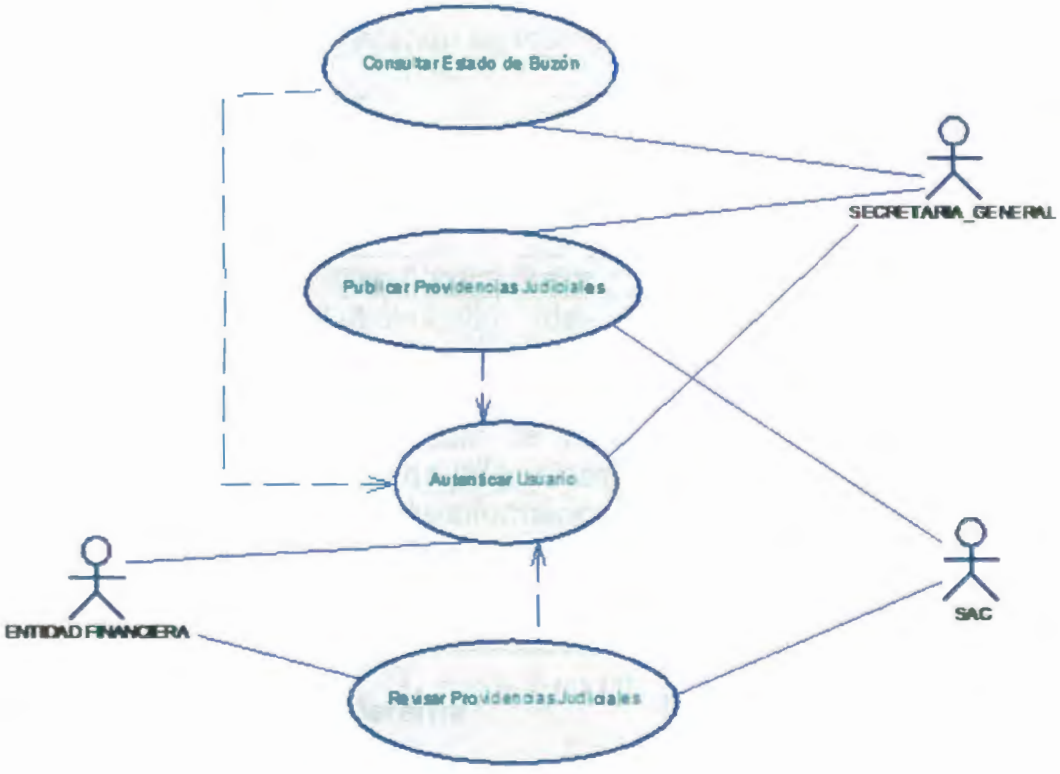
5.1. Requisitos de almacenamiento de información

RI-01	***Almacenamiento de documentos digitalizados***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna
Objetivos Asociados	• OBJ-01 - Mejorar el proceso de notificación. • OBJ-02 - Controlar el repudio de la información.
Requisitos Asociados	--
Descripción	Realizar el almacenamiento en archivos comprimidos de las providencias y circulares emitidas por Secretaría General.
Datos Específicos	Existe un volumen inicial de número de registros de 4.400 tomando como referencia los documentos impresos del año 2007. El crecimiento anual estimado de documentos es del 50% debido a la eliminación de ciertos conceptos para la emisión de Providencias Judiciales.
Intervalo temporal	Pasado y presente
Importancia	Vital
Urgencia	Inmediatamente
Estado	En construcción
Estabilidad	Alta
Comentarios	--

5.2. Requisitos funcionales

5.2.1. Diagrama de casos de uso

Object-Oriented Model	
Model	Requisitos Funcionales
Package	
Diagram	Casos de Uso (Sistema de Providencias Judiciales)
Author	wcargua Date 09/06/2008
Version	



5.2.2. Definición de Actores

ACT-01	***Unidad de Secretaría General de la SBS***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna
Descripción	Este actor representa al personal de la Unidad de la SBS

	que tiene el rol de administrador de la nueva aplicación de Providencias Judiciales.
Comentarios	--

ACT-02	***Entidad Controlada por la SBS***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna
Descripción	Este actor representa al personal de la entidad controlada por la SBS que tiene el rol de consulta en la nueva aplicación de Providencias Judiciales.
Comentarios	--

ACT-03	***Sistema de Catastros de la SBS***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• Subdirección de Desarrollo y Aplicaciones Tecnológicas
Descripción	Este actor representa al Sistema de Catastros de la SBS mediante el cual se realiza la administración de la información de las entidades controladas. Este actor es un sistema de información que interactúa directamente con los procesos definidos para la nueva aplicación de Providencias Judiciales.
Comentarios	--

5.2.3. Casos de uso del Sistema

RF-01	***Publicar Providencias Judiciales***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna
Objetivos Asociados	• OBJ-01 - Mejorar el proceso de notificación.
Requisitos Asociados	RI-01, RF-02
Descripción	El usuario delegado de Secretaría General en este caso de uso realiza el ingreso diario de la información de los

	documentos de Providencias Judiciales	
Precondición	• Que existan documentos de Providencias Judiciales digitalizadas • El usuario de Secretaría general debe tener asignados los permisos para realizar el ingreso de información • El usuario de Secretaría General debe estar autenticado en el Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario de Secretaría General ingresa a la opción de nuevo documento
	2	El usuario de Secretaría General ingresa o modifica los datos correspondientes a un documento de providencia judicial digitalizada y escoge el archivo digitalizado
	3	El usuario de Secretaría General ingresa o modifica las entidades controladas a las que va dirigida la Providencia Judicial.
	4	El sistema realiza las validaciones de los datos de la providencia judicial
	5	Si el sistema validó los datos de la providencia judicial, almacena la información y devuelve un mensaje de éxito al usuario, caso contrario permite al usuario repetir los pasos 2-3 hasta que los datos sean válidos
	6	El sistema luego de almacenar la información envía una notificación vía correo electrónico, para recordar a las entidades controladas que revisen la nueva publicación de una providencia judicial
Postcondición	El documento de Providencia Judicial está disponible en el Sistema para ser consultado por las entidades controladas	
Excepciones	**Paso**	**Acción**
	4	Si los datos de la providencia judicial son erróneos, el sistema envía un mensaje de error al usuario para que realice la corrección respectiva y poder continuar con el almacenamiento de la providencia
Rendimiento	**Paso**	**Cuota de tiempo**
	1	5 segundos
	2	10 segundos

	3	5 segundos
	4	5 segundos
	5	15 segundos por cada iteración
	6	10 segundos
Frecuencia esperada	Número de veces: 10 / 1 día	
Importancia	Vital	
Urgencia	Normal	
Estado	En construcción	
Estabilidad	Media	
Comentarios	--	

RF-02	***Autenticar Usuario***	
Versión	1.0	
Autores	LSI. William Cargua Freire	
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna	
Objetivos Asociados	• OBJ-02 - Controlar el repudio de la información.	
Requisitos Asociados	RF-01, RF-03, RF-04	
Descripción	Este caso de uso se debe realizar al iniciar los casos de uso: Publicar Providencias Judiciales, Revisar Providencias Judiciales, Consultar estado de revisión del buzón	
Precondición	Existe disponibilidad del sistema y el usuario se encuentra registrado en el sistema con su respectivo perfil de acceso	
Secuencia Normal	**Paso**	**Acción**
	1	El sistema solicita al usuario su nombre de usuario y clave de acceso
	2	El usuario proporciona al sistema su nombre corto y su clave de acceso
	3	El sistema comprueba si el nombre corto del usuario y la clave de acceso son correctas
	4	Si el nombre corto del usuario y la clave no son correctas, el sistema permite al usuario repetir el intento (pasos 1-3) hasta un máximo de tres veces
	5	Si el nombre corto del usuario y la clave son correctas, el sistema permite el acceso al usuario

Postcondición	El usuario puede acceder al sistema con los permisos de acceso de su respectivo perfil	
Excepciones	**Paso**	**Acción**
	4	Si el usuario ha intentando tres veces acceder sin éxito, el sistema rechaza el acceso del usuario, a continuación este caso de uso termina
Rendimiento	**Paso**	**Cuota de tiempo**
	1	5 segundos
	2	5 segundos
	3	20 segundos
	4	30 segundos por cada iteración
	5	5 segundos
Frecuencia esperada	Número de veces: 240 / 1 día	
Importancia	Vital	
Urgencia	Inmediatamente	
Estado	En construcción	
Estabilidad	Alta	
Comentarios	--	

RF-03	***Revisar Providencias Judiciales***	
Versión	1.0	
Autores	LSI. William Cargua Freire	
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna	
Objetivos Asociados	• OBJ-01 - Mejorar el proceso de notificación. • OBJ-02 - Controlar el repudio de la información.	
Requisitos Asociados	RF-02	
Descripción	El usuario delegado de cada entidad controlada en este caso de uso realiza la descarga de los documentos de Providencias Judiciales	
Precondición	• El usuario de la entidad controlada debe tener asignados los permisos para realizar la descarga de información de documentos de Providencias judiciales • El usuario de la Entidad controlada debe estar autenticado en el Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario de la entidad controlada escoge el

		documento de Providencia Judicial que desea descargar
	2	El sistema registra el acceso del usuario de la entidad controlada por cada documento de providencia judicial
	3	El sistema envía un mensaje de éxito de la descarga al usuario
Postcondición	El documento de providencia judicial es descargado por el usuario de la entidad controlada	
Excepciones	**Paso**	**Acción**
	1	Si no existen publicaciones, el sistema envía un mensaje informando al usuario de la entidad controlada y este caso de uso termina
Rendimiento	**Paso**	**Cuota de tiempo**
	1	5 segundos
	2	10 segundos
Frecuencia esperada	Número de veces: 240 / 1 día	
Importancia	Vital	
Urgencia	Normal	
Estado	En construcción	
Estabilidad	Media	
Comentarios	--	

RF-04	***Consultar Estado de Revisión del Buzón***
Versión	1.0
Autores	LSI. William Cargua Freire
Fuentes	• LCDA. Letty Suárez Carrasco • LCDO. Pablo Cobo Luna
Objetivos Asociados	• OBJ-02 - Controlar el repudio de la información.
Requisitos Asociados	RF-02
Descripción	El usuario delegado de Secretaría General en este caso de uso realiza la revisión de la descarga de los documentos de Providencias Judiciales
Precondición	• El usuario de Secretaría General debe tener asignados los permisos para realizar la revisión de las descargas de los documentos de Providencias judiciales

	• El usuario de Secretaría General debe estar autenticado en el Sistema	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario de Secretaría General escoge el documento de providencia judicial para la revisión de las descargas realizadas
	2	El sistema genera un listado de los usuarios de entidades controladas con la fecha y hora de acceso.
	3	El sistema presenta el listado generado en el paso 2 y lo presenta al usuario
Postcondición	El usuario de Secretaría General obtiene el listado de los usuarios que han realizado descargas del documento de providencia judicial	
Rendimiento	**Paso**	**Cuota de tiempo**
	1	5 segundos
	2	1 segundo
	3	1 segundo
Frecuencia esperada	Número de veces: 5 / 1 día	
Importancia	Importante	
Urgencia	Normal	
Estado	En construcción	
Estabilidad	Media	
Comentarios	--	

6. Matriz de rastreabilidad objetivos/requisitos

	OBJ-01	***OBJ-02***
RI-01	•	•
RF-01	•	
RF-02		•
RF-03	•	•
RF-04		•

"ANEXO-13-Documento de Estudio de Factibilidad del prototipo"

ESTUDIO DE FACTIBILIDAD

Proyecto de automatización: Sistema de Providencias Judiciales

1. Identificación del área solicitante

Fecha del estudio (dd/mm/yyyy): 11/06/2008

Área usuaria de la aplicación: SECRETARÍA GENERAL

Nombre del titular del área usuaria: LCDO. PABLO COBO LUNA

Usuario contraparte: LCDA. LETTY SUÁREZ CARRASCO

2. Especificaciones del Proyecto

1.1. Nombre del Sistema

SISTEMA DE PROVIDENCIAS JUDICIALES

1.2. Siglas de identificación

SPJ

1.3. Versión

1.0

1.4. Antecedentes

La Superintendencia de Bancos y Seguros mediante la unidad de Secretaria General mantiene un proceso manual para realizar la notificación de las providencias judiciales a las entidades controladas.

Para iniciar este proceso de notificación, secretaria general recepta las providencias judiciales debidamente legalizadas mediante oficios o con la descripción de su alcance en las

localidades de la Superintendencia de Bancos y Seguros. Cabe señalar que esta recepción de documentos se la realiza con una frecuencia diaria.

Luego una persona encargada en Secretaria General realiza la redacción de una circular en donde se indica cuáles es o son las providencias judiciales recibidas y que luego serán objeto de revisión por parte de las entidades financieras controladas.

Una vez terminada la redacción de la circular una persona encargada de Secretaria General realiza la reproducción de copias fotostáticas de la o las providencias recibidas en un número aproximado de 220 copias según el número de providencias que se reciben diariamente. Después la persona encargada organiza la o las providencias fotocopiadas con la circular respectiva y las ajunta en un sobre para su posterior envío.

Los sobres con los documentos son enviados mediante una empresa externa de transporte y entrega, dirigidos a todas las entidades financieras controladas.

1.5. Objetivos del proyecto

- Simplificar el proceso de notificación, es decir proporcionar información confiable, oportuna y segura en la entrega de las providencias judiciales.

- Ejercer un mejor control sobre el repudio de la recepción de las providencias por parte de las entidades financieras controladas.

1.6. Principales problemas a resolver

- Un inconveniente que tienen en Secretaria General se origina en una acción posterior a la notificación de las providencias judiciales, la cual se da por el motivo del repudio de la información por ciertas entidades financieras, es decir, que las entidades en muchas ocasiones indican que la acción de recepción de las providencias por parte de ellas nunca se llevó a cabo. Es entonces que la Secretaria General solicita a la empresa externa de transporte que le entregue los recibos

de recepción firmados por la entidad financiera controlada, para luego poder fundamentar su notificación en conjunto con el archivo interno de la Superintendencia de Bancos y Seguros.

- Falta de oportunidad de información para la toma de decisiones en lo que se refiere a la confirmación de la recepción y revisión oportuna de las providencias por parte de las entidades controladas.

- Demasiados recursos empleados para poder realizar las notificaciones debido a procesos manuales de fotocopiado y al consumo en exceso de papel.

- La falta de control sobre la recepción, revisión y acceso a las providencias judiciales por parte de las entidades financieras controladas.

1.7. Restricciones, limitantes

- Destinar los recursos tecnológicos de forma oportuna para iniciar la etapa de construcción del software de envío electrónico de providencias judiciales.

- Solventar los siguientes requerimientos normativos y de gestión de información:

  a. Incluir toda la información posible que intervenga en el proceso de notificación de providencias, la que luego servirá para el mejoramiento continuo de las operaciones del software aplicativo.

  b. Documentar un procedimiento funcional dentro de Secretaria General con las especificaciones detalladas de cómo se va a realizar la nueva forma de notificación de las providencias a las entidades controladas, incluyendo la utilización del nuevo software aplicativo.

  c. El otorgamiento del número de licencias de procesamiento.

1.8. Beneficios que se obtendrán

- Reducción de recursos como papel, componentes de los equipos de fotocopiado y materiales de oficina.
- Control de recepción y revisión de las providencias judiciales por parte de las entidades controladas.
- Reducción de procesos manuales y operativos para realizar las notificaciones a las entidades.
- Ser más eficientes en la atención que la Superintendencia de Bancos brinda en las notificaciones de la providencias.
- El almacenamiento organizado de las providencias junto con las circulares conformadas en un paquete de comunicación electrónico, mediante la cual se reducirá la organización de los documentos impresos.

1.9. Usuarios principales, roles u opciones del sistema

Perfiles y opciones:

	PERFILES	
DESCRIPCIÓN DE LA OPCIÓN	**Administrador**	**Entidad Externa**
1. ADMINISTRACIÓN	V	I
1.1. Publicación	V	I
2. SERVICIOS	V	V
2.1. Descarga	V	V

Usuarios:

Descripción	# de Usuarios
SBS	10
ENTIDADES	240

1.10. Infraestructura de hardware y software necesaria para la solución

Hardware:

Nombre	Descripción	Nuevo/Existente
Servidor de BDD	2 HP blade AMD dual core 2.6 GHz, 28 GB RAM por blade, 1200 GB internos en 4 discos, 2 por cada blade.	Existente
Servidor de Componentes	1 HP blade AMD dual core 2.6 GHz, 14 GB RAM por blade y 300 GB internos en 2 discos.	Existente
PC usuario	Pentium IV 1.7 GHz, 512 MB RAM.	Existente

Software:

Nombre del producto	Modelo/Versión	Nuevo/Existente
Base de datos Oracle	10g, ver 10.2.0 o superior	Existente
Oracle Aplication Server	10g, ver 10.1.2 o superior	Existente
Oracle JDeveloper	ver 10.1.2 o superior	Existente
ADF Business Components	ver 10.1.3	Existente
Java Platform (J2EE)	ver 1.4	Existente
Power Designer	ver 9.5 o superior	Existente

1.11. Tecnologías de desarrollo

Para cumplir con el estándar de desarrollo de las aplicaciones WEB de la institución, se utilizará ADF Business Component a través de la herramienta de desarrollo JDeveloper 10.1.2 o superior.

1.12. Tiempo aproximado de desarrollo

Antes de medir el tiempo para el desarrollo del sistema se muestra un cronograma detallado de las tareas a realizarse en cuento a la factibilidad del desarrollo de la solución.

	Nombre de tarea	Duración	Comienzo	Fin
1	SISTEMA DE PROVIDENCIAS JUDICIALES	34 días	mié 11/06/08	lun 28/07/08
2	ANALISIS	6 días	mié 11/06/08	mié 18/06/08
3	DISEÑO	10 días	jue 19/06/08	mié 02/07/08
4	PROGRAMACION	15 días	jue 03/07/08	mié 23/07/08
5	IMPLEMENTACION	3 días	jue 24/07/08	lun 28/07/08

Para el cálculo del tiempo de desarrollo del proyecto se ha utilizado la métrica del número de pantallas y su dificultad.

Métricas				
Dificultad	**Opción**	**# de Pantallas**	**# Días desarrollo**	**Subtotal (días)**
Alta	Procesos	1	12	12
Media		0	0	0
Baja		1	3	3
Subtotal Pantallas:		**2**	**15**	
		Total(días)		**15**

1.13. Presupuesto Referencial

Por tratarse de un proyecto piloto el presupuesto referencial se limita a los costos de suministros.

2. Descripción conceptual de la aplicación

La nueva aplicación utilizará las facilidades de acceso que brinda actualmente la tecnología Web, proporcionando a los usuarios que realizan la revisión de Providencias Judiciales, la capacidad de consultar esta información en tiempo casi real. De esta manera el tiempo de la distribución de las providencias en papel sería eliminado, con los correspondientes costos que esta situación generaba.

A través de las seguridades que brinda el portal de la SBS, se podrá establecer mejor control de los usuarios que accedan a la información y de esta manera evitar cualquier problema de infiltraciones de usuarios ajenos a la entidad y de los posibles repudios de información.

Puesto que este nuevo sistema será desarrollado en la tecnología de tres capas, el procesamiento de los datos será distribuido entre el servidor de aplicaciones y el servidor de base de datos por lo que la velocidad de respuesta hacia los usuarios será muy buena.

3. Conclusión sobre la factibilidad del proyecto

Factibilidad Económica:

El proyecto sólo incurriría en los costos de suministros de oficina y de gatos de servicios básicos, por lo cual el proyecto es factible económicamente.

Factibilidad Técnica:

Puesto que se cuenta con el conocimiento y la experiencia del técnico que realizará el desarrollo de este proyecto, la parte técnica está garantizada.

De igual manera la DNRT cuenta entre sus funcionarios con las personas técnicas necesarias tanto para administración de base de datos y comunicaciones que es otra garantía para el desarrollo del proyecto.

Así mismo, se cuenta con toda la infraestructura tecnológica tanto en software como en hardware para que la aplicación sea implantada.

Factibilidad Organizacional:

La organización contará con un sistema más con todas las seguridades del portal de la SBS y las facilidades de acceso y uso que brindan las aplicaciones WEB actuales.

Por lo expuesto anteriormente se concluye que el desarrollo de la aplicación Web de Providencias Judiciales es factible de realizar.

Manifestamos nuestra conformidad con lo arriba expuesto.

Atentamente,

LCDO. Pablo Cobo Luna
SECRETARIO GENERAL

LCDA. Letty Suárez Carrasco
SECRETARÍA GENERAL

LSI. William Cargua Freire
Desarrollo de Aplicaciones

"ANEXO-14-Diagramas para el desarrollo del prototipo"

A14.1. Diagrama de Clases de la aplicación: "Providencias Judiciales"

Object-Oriented Model	
Model: Modelado Conceptual	
Package:	
Diagram: Diagrama de Clases (Sistema de Providencias Judiciales)	
Author: wcargua	Date : 19/06/2008
Version :	

SpjInterfaceWebInst

+	usr_usuarioapp	: String	
+	nom_usuarioapp	: String	
+	<<Getter>>	getNomUsuarioapp ()	: String
+	<<Getter>>	getUsrUsuarioapp ()	: String
+		ingresarEAI ()	: int
+		entrarPJU ()	: int

SpjInterfacePju

+	cod_sistema	: String	
+	txt_sistema	: String	
+	<<Getter>>	getCodSistema ()	: String
+	<<Getter>>	getTxtSistema ()	: String
+		entrarPJU ()	: int
+		crearBuzon ()	: int
+		getContenidoBuzon ()	: java.lang.Object

SpjBuzon

-	cod_circular	: String	
-	fec_recepcion	: Date	
-	fec_paquete	: Date	
-	txt_contenido	: String	
-	nom_paquete	: String	
-	obj_documentos	: SpjDocumentos	
+	<<Constructor>>	SpjBuzon ()	
#	<<Destructor>>	finalize ()	: void
+	<<Getter>>	getCodCircular ()	: String
+	<<Setter>>	setCodCircular (String newCod_circular)	: void
+	<<Getter>>	getFecRecepcion ()	: Date
+	<<Setter>>	setFecRecepcion (Date newFec_recepcion)	: void
+	<<Getter>>	getFecPaquete ()	: Date
+	<<Setter>>	setFecPaquete (Date newFec_paquete)	: void
+	<<Getter>>	getTxtContenido ()	: String
+	<<Setter>>	setTxtContenido (String newTxt_contenido)	: void
+	<<Getter>>	getNomPaquete ()	: String
+	<<Setter>>	setNomPaquete (String newNom_paquete)	: void
+		crearBuzon ()	: String
+		actualizaPaquete ()	: String
+		eliminaPaquete ()	: String
+		tgrCreaBuzon ()	: int
+		tgrEliminaPaquete ()	: int
+		tgrActualizaPaquete ()	: int
+	<<Getter>>	getObjDocumentos ()	: SpjDocumentos
+		getContenidoBuzon ()	: java.lang.Object

1..1 — 1..*

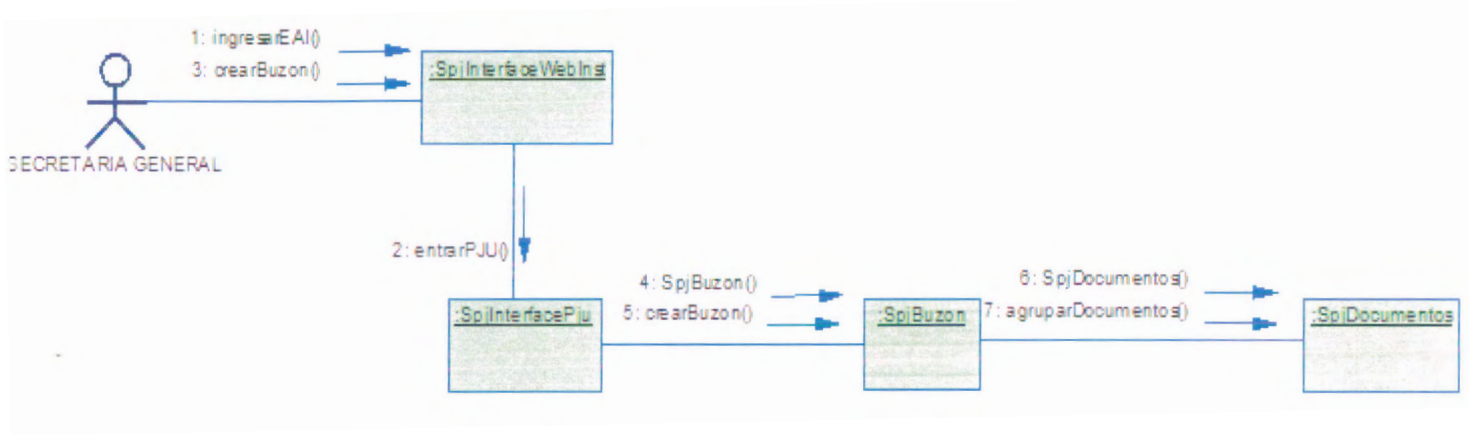
SpjDocumentos

-	cod_documento	: String	
-	cod_institucion	: String	
-	txt_documento	: String	
-	sts_revision	: String	
+	<<Constructor>>	SpjDocumentos ()	
#	<<Destructor>>	finalize ()	: void
+	<<Getter>>	getCodDocumento ()	: String
+	<<Setter>>	setCodDocumento (String newCod_documento)	: void
+	<<Getter>>	getCodInstitucion ()	: String
+	<<Setter>>	setCodInstitucion (String newCod_institucion)	: void
+	<<Getter>>	getTxtDocumento ()	: String
+	<<Setter>>	setTxtDocumento (String newTxt_documento)	: void
+	<<Getter>>	getStsRevision ()	: String
+	<<Setter>>	setStsRevision (String newSts_revision)	: void
+		tgrActualiza ()	: int
+		aguparDocumentos ()	: java.lang.Object
+		getDocumentos ()	: java.lang.Object

A14.2. Diagramas de Colaboración de la aplicación: "Providencias Judiciales"

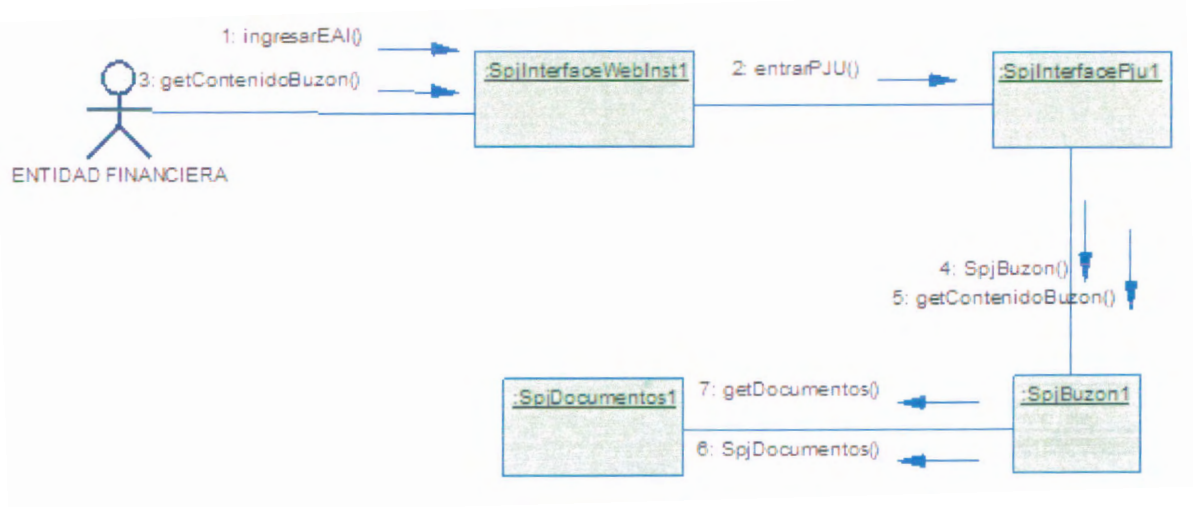
A.14.2.1. Diagrama de Colaboración del Proceso: Publicación de Documentos

Object-Oriented Model	
Model	Diagramas de Interacción
Package	
Diagram: Diagrama de Colaboracion (Publicación de documentos)	
Author: wcargua	Date: 01/07/2008
Version	

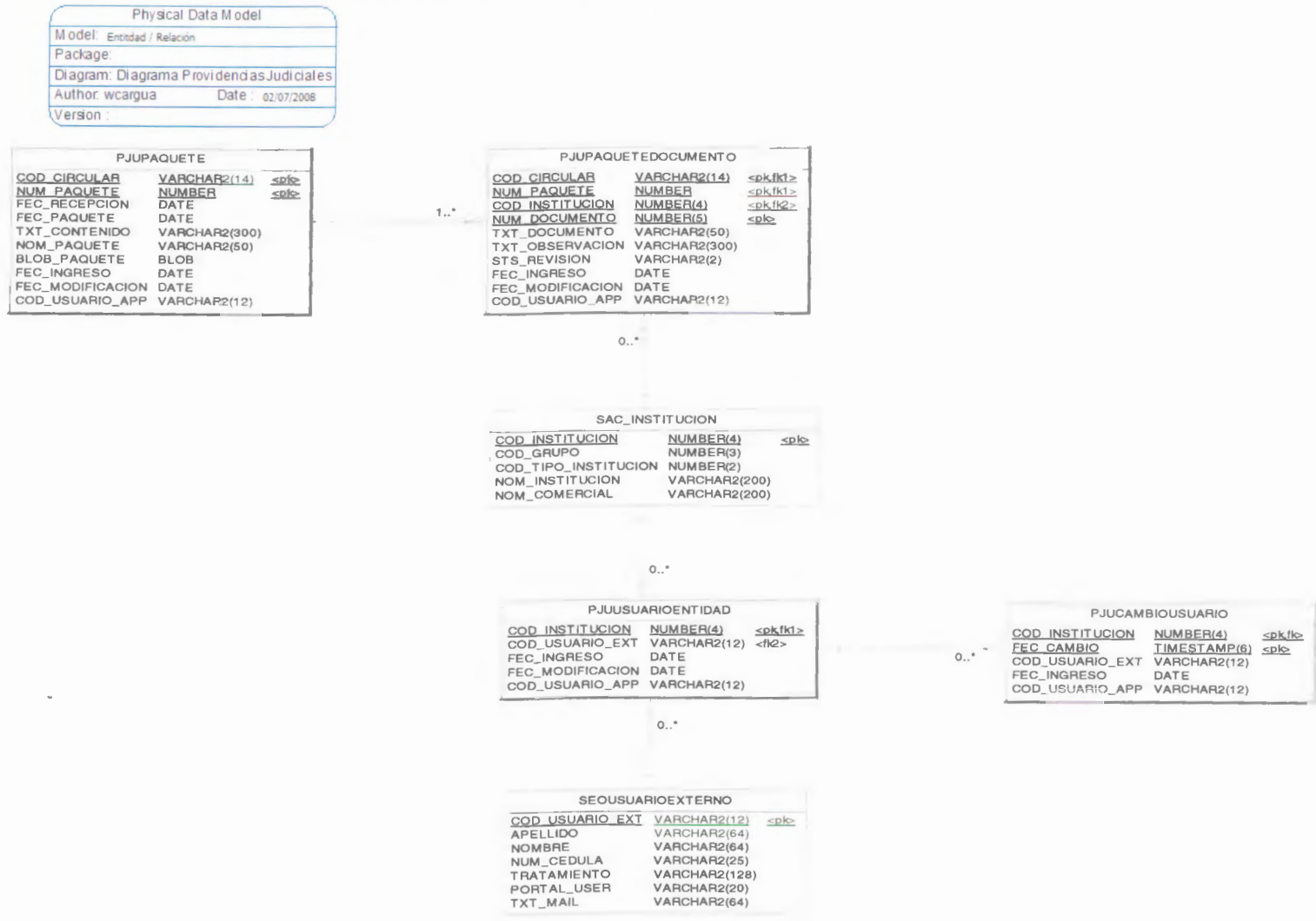


A.14.2.2. Diagrama de Colaboración del Proceso: Revisión de Documentos

Object-Oriented Model	
Model: Diagramas de interacción	
Package:	
Diagram: Diagrama de Colaboracion (Revisión de documentos)	
Author: wcargua	Date: 27/06/2008
Version:	



A.14.3. Diagrama Entidad/Relación de la aplicación: "Providencias Judiciales"




MSIG
MAESTRÍA EN SISTEMAS DE
INFORMACIÓN GERENCIAL
ESCUELA SUPERIOR POLITECNICA DEL LITORAL
ESPOL

GLOSARIO

GLOSARIO

- A -

Accesibilidad Web: La accesibilidad web hace referencia a la capacidad de acceso a un sitio Web por todo tipo de usuarios, independientemente de sus discapacidades o su contexto de navegación, de modo que los usuarios serán capaces de percibir, entender, navegar e interactuar con dicho sitio.

Algoritmo: Procedimiento o conjunto de procedimientos que describen una asociación de datos lógicos destinados a la resolución de un problema. Los algoritmos permiten automatizar tareas.

Análisis: Proceso que genera el modelo conceptual de un dominio de interés. Cuando se desea construir un sistema (sea de software o no) para un dominio, el análisis también "captura" los requisitos que el sistema debe cumplir.

API (Application Program Interface): Programa de interface de una aplicación, especificación de una función que realiza un llamado a una función que realiza un servicio.

Aplicación: Programa de computadora orientado a automatizar alguna actividad o conjunto de actividades de un proceso específico.

Archivos log: Archivo de texto que almacena generalmente datos sobre procesos determinados. Para entendernos, es como el "diario" de algunos programas donde se graban todas las operaciones que realizan, para posteriormente abrirlos y ver qué es lo que ha sucedido en cada momento.

- B -

Base de datos: Una base de datos es un conjunto de archivos interrelacionados que se crea y se gestiona mediante un sistema de gestión de bases de datos (DBMS).

Base de datos relacional: Una base de datos relacional es una colección de datos organizados como un conjunto de tablas cuyos datos pueden extraerse o reagruparse de muchas formas diferentes sin tener que reorganizarlas.

Batch: En informática, un BATCH es un programa que se ejecuta de forma independiente sin la interacción del usuario. Un ejemplo de archivo Batch puede ser el AUTOEXEC.BAT de los antiguos sistemas basados en DOS.

Browser, o navegador: Una categoría genérica utilizada para describir aplicaciones de software que permiten a los usuarios visualizar archivos de tipo web.

Business Intelligence (BI): La Inteligencia de Negocios o Business Intelligence (BI) se puede definir como el proceso de analizar los datos acumulados en la empresa y extraer conocimiento de ellos.

- C -

Cascading Style Sheet (CSS), u hojas de estilo en cascada: Formato para especificar la presentación del contenido HTML o XML, en base a los nombres de etiquetas y atributos, sus posiciones, etc.

Caso de uso (UC): Un caso de uso es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema.

Clase: En un modelo de dominio, define un conjunto de propiedades compartido por un determinado grupo de entidades (cosas materiales, conceptos, ideas, sucesos); es una abstracción de los objetos similares de un dominio.

Clic (click): Acción mediante la cual el hipervínculo situado justamente debajo del cursor queda activado; para ello se ha de presionar sobre el hipervínculo con el botón situado a la izquierda del ratón.

Cliente: Cliente o 'programa cliente' es aquel programa que permite conectarse a un determinado sistema, servicio o red.

Cliente-Servidor: Se denomina así al binomio consistente en un programa cliente que consigue datos de otro llamado servidor sin tener que estar obligatoriamente ubicados en el mismo ordenador. Esta técnica de consulta 'remota' se utiliza frecuentemente en redes como 'Internet'.

Comercio electrónico (e-commerce): Intercambio de bienes y servicios realizado a través de herramientas asociadas con las tecnologías de la

información y las comunicaciones, habitualmente con el soporte de las plataformas y protocolos estandarizados.

Criptografía: Ciencia dedicada al estudio de técnicas capaces de conferir seguridad a los datos. El cifrado es fundamental a la hora de enviar datos a través de las redes de telecomunicaciones con el fin de conservar su privacidad.

- D -

DBMS: Un sistema de gestión de bases de datos (DBMS). También suele recibir el nombre de gestor de bases de datos. Se trata de un programa que permite que uno o más usuarios informáticos puedan crear y acceder al contenido de una base de datos. El DBMS gestiona las solicitudes de los usuarios (y las solicitudes de otros programas) para que no tengan que preocuparse del lugar físico en el que están almacenados los datos y, en un sistema multiusuario, de quién más puede acceder a los datos.

DHTML (HTML Dinámico): Es una técnica para crear sitios Web interactivos mediante el uso de una combinación de HTML estático, algún lenguaje de programación de ejecución en el navegador como por ejemplo javascript y hojas de estilo (CSS).

Diseño: Proceso que usa los resultados del análisis para generar una especificación de la implementación de un sistema o producto. Los diseños suelen usar dibujos, iconos, modelos o planos. A menudo, la palabra diseño se utiliza también para referirse a la descripción lógica del funcionamiento del sistema o producto.

Dominio: Parte del mundo real bajo estudio. Cuando consideramos sistemas de software, el dominio es el campo o ámbito para el cual se construye el sistema. Por ejemplo, una aplicación de contabilidad cae dentro del dominio financiero.

- E -

Escalabilidad: La posibilidad de un sistema de incrementar su capacidad para adaptarse a un incremento de sus exigencias.

Etiqueta (tag): Instrucción que se escribe al elaborar un documento HTML. El conjunto de las etiquetas que aparecen en una página son interpretadas

por el programa navegador para visualizar dicha página de forma adecuada en una pantalla.

- F -

Facilidad de uso: La facilidad de uso está en relación directa con la eficiencia o efectividad, medida como velocidad o cantidad de posibles errores.

Una herramienta muy fácil de usar permitirá a su usuario efectuar más operaciones por unidad de tiempo (o menor tiempo para la misma operación) y disminuirá la probabilidad de que ocurran errores.

Factibilidad: Es la disponibilidad de los recursos necesarios para llevar a cabo los objetivos o metas señaladas, sirve para recopilar datos relevantes sobre el desarrollo de un proyecto y en base a ello tomar la mejor decisión.

Flujo de trabajo: Secuencia de pasos que son procesados por personas o automáticamente por el sistema.

- H -

Heurística: La heurística es un conjunto de reglas a seguir basadas en la experiencia (información empírica) para la validación de un procedimiento.

HTML: HyperText Markup Language, o lenguaje de marcación de hipertexto. El conjunto de códigos o símbolos de marcación insertados en un archivo para su visualización en una página de un navegador de Internet, cuya función es dar instrucciones al navegador sobre cómo ha de ver el usuario las palabras y las imágenes de la web.

HTTP: HyperText Transfer Protocol, o protocolo de transferencia de hipertexto. Es el protocolo subyacente utilizado por la World Wide Web. El HTTP define la forma en que se formatean y se transmiten los mensajes.

- I -

Ingeniería del Software: Disciplina cuyo propósito es la producción de software libre de fallos, dentro del plazo previsto, cumpliendo el presupuesto inicial, y que satisfaga las necesidades del usuario o cliente.

Instancia: Materialización de un objeto durante el tiempo de ejecución de un programa. En términos computacionales, una instancia corresponde directamente a un bloque contiguo de memoria, que comienza en una dirección de memoria y ocupa cierto espacio.

Integridad: La habilidad de determinar que la información recibida es la misma que la información enviada.

- J -

Java: Java y JSP son tecnologías interrelacionadas y tienen nombres similares, lo que puede resultar confuso. Java es un lenguaje de programación (como Visual Basic o C++).

JavaScript: JavaScript es un lenguaje de programación utilizado en las páginas HTML para incorporar interactividad y funcionalidad adicional, normalmente en el entorno del lado del cliente.

JSP: JSP significa Java Server Pages, y en todos los sentidos puede interpretarse como la alternativa de Java a las páginas ASP (Active Server Pages) de Microsoft.

- M -

Mantenimiento correctivo: Conjunto de medidas de mantenimiento que no han sido generadas a partir de un plan previo.

Mantenimiento preventivo: Medidas de mantenimiento que se realizan periódicamente, generadas a partir de un plan definido previamente.

Mensaje: Estímulo enviado a un objeto con un nombre y unos argumentos adecuados, que provoca que el objeto comience cierto comportamiento al activarse la operación asociada al mensaje. Los mensajes modifican el estado del objeto o devuelven información sobre su estado.

Por ejemplo, un mensaje devolver Altura enviado a un objeto Persona devuelve la altura del objeto.

Meta Tags, o metaetiquetas: Se trata de una etiqueta HTML que identifica los contenidos de una web. Las metaetiquetas contienen información como una descripción general de la página, palabras clave para los motores de

búsqueda e información sobre los derechos de propiedad. Ver también Metadatos.

Metadatos: Meta es un prefijo que en su utilización informática significa "definición o descripción subyacente". Metadatos es, por lo tanto, una definición o descripción de datos. Ver también Meta Tags.

Metodología: Colección de técnicas repetibles para resolver una familia de problemas. Por ejemplo, un libro de bricolaje contiene una metodología para hacer, en la propia vivienda, obras de carpintería, fontanería y electricidad. En software, el análisis y el diseño suelen realizarse siguiendo una determinada metodología.

Modelo: Esquema teórico de un sistema o de una realidad compleja, que se elabora para facilitar su comprensión y el estudio de su comportamiento. Un modelo suele representarse mediante diagramas más los textos, notaciones o aclaraciones necesarias para entenderlos.

Modelo de Software: Representación de un componente de software (una clase o un módulo, por ejemplo) o un sistema de software. Los modelos de software suelen representarse en UML.

- O -

Objeto: En un modelo de dominio, abstracción de alguna entidad presente en el dominio de interés.

Operación: Descripción de la habilidad de un objeto para responder a un mensaje, así como de los requisitos de éste.

Oracle: El mayor proveedor mundial de software de desarrollo de bases de datos y aplicaciones.

Orientación a Objetos (OO): Metodología para desarrollar sistemas mediante clases y objetos.

- P -

Paradigma: Estrategia o punto de vista para realizar tareas o resolver problemas. Marco conceptual.

PDF: Portable Document Format, o formato de documento portátil. Tipo de archivo que utiliza el lenguaje para impresoras PostScript y es extremadamente portátil entre diferentes plataformas informáticas.

Portal: Sitio web cuyo objetivo es ofrecer al usuario, de forma fácil e integrada, el acceso a una serie de recursos y de servicios, entre los que suelen encontrarse buscadores, foros, compra electrónica, etc.

Programación Orientada a Objetos (POO): Metodología de programación basada en objetos y en el envío de mensajes entre éstos.

Los fundamentos de la POO son los de la OO.

Prototipo: Un prototipo es un modelo de representación, demostración o simulación de un sistema planificado, probablemente incluyendo su interfaz y su funcionalidad.

Proyecto: Designio, propósito o pensamiento de hacer algo. Previsión, ordenamiento o premeditación que se hace para realizar o ejecutar una obra u operación.

- R -

Relación: Abstracción de un conjunto de interrelaciones semánticas concretas que se dan sistemáticamente entre distintos tipos de objetos.

Requisito: Descripción de lo que debe hacer un sistema o producto (requisito funcional) o de cómo debe implementarse (requisito no funcional o técnico).

RFC (Solicitud de comentarios, Request For Comments): Documentos a través de los cuales se proponen y efectúan cambios en Internet, en general con orientación técnica.

- S -

Servicio Informático: Conjunto de actividades (planeamiento, análisis, diseño, programación, operación, entrada de datos, autoedición, bases de

datos, etc.) asociadas al manejo automatizado de la información que satisfacen las necesidades de los usuarios de éste recurso.

Servidor (server): Sistema que proporciona servicios relacionados con un recurso en particular a los usuarios; por ejemplo, servidor de archivos, servidor de nombres o servidor de correo electrónico, ya sea en una red interna o externa. En Internet este término se utiliza muy a menudo para designar a aquellos sistemas que proporcionan información a los usuarios de la red.

Servidor Web (Web server): Aplicación que sirve archivos de un sitio web a petición de los usuarios. Se llama así también a la máquina conectada a la red en la que están almacenadas físicamente las páginas que componen un sitio.

Sistema de Información: Se denomina Sistema de Información al conjunto de procedimientos manuales y/o automatizados que están orientados a proporcionar información para la toma de decisiones.

Sitio Web (Website): Punto de la red con una dirección única y al que pueden ingresar los usuarios para obtener información. También llamado site o sitio, normalmente un sitio web dispone de un conjunto de páginas organizadas a partir de una "home page" o página principal, e integra archivos de varios tipos, tales como sonidos, fotografías, o aplicaciones interactivas de consulta (formularios). Esas páginas se cohesionan normalmente por la pertenencia a un tipo de contenidos o a una organización o empresa.

SQL: Structured Query Language, o lenguaje de consulta estructurado. Lenguaje utilizado para interrogar y procesar datos en una base de datos relacional. Todos los sistemas de bases de datos diseñados para entornos de cliente/servidor soportan SQL.

- T -

Tabla de base de datos: Es el conjunto de Registros que conforman un tipo de información sobre diversos elementos o temas. Un grupo de tablas conforman una Base de Datos.

Taxonomía: Ciencia que trata de los principios, métodos y fines de la clasificación.

En el ámbito de la arquitectura de la información de portales web, puede ser definida como el proceso general de organización de contenidos en diferentes categorías orientadas hacia su mejor navegación, organización y búsqueda.

TCP/IP (Protocolo de transmisión y control/Protocolo de Internet): Es el protocolo básico de comunicación utilizado en Internet.

Tiempo de respuesta (Response time): Lapso de tiempo que transcurre entre la petición de información a la red por parte de un usuario y su recepción por éste. Este tiempo de respuesta depende de muchas variables, desde la propia computadora hasta las características de las telecomunicaciones del país donde habita.

- U -

UML: Lenguaje gráfico que se usa principalmente para visualizar, especificar, construir y documentar componentes de software (una clase de Java, por ejemplo) y sistemas de software (una aplicación de gestión empresarial, p. ej.). Aunque UML se ha convertido en el lenguaje estándar para los modelos de software OO, es un lenguaje de modelado de propósito general y puede usarse para modelar sistemas que no son de OO ni de software (por ejemplo, empresas).

URL: Siglas inglesas de Uniform Resource Locator. Es la dirección de un archivo (o recurso) al que se puede acceder en Internet - p. ej: http://www.superban.gov.ec

Usuario final: La persona que utiliza un producto determinado, es decir, el cliente final, en este caso el cliente principal de la SBS son las unidades que la integran.

- W -

Web: El término se utiliza para definir el universo del World Wide Web, los sitios, la información y los servicios de la "teleraña". Han existido diversos intentos de imponer una traducción adecuada al español, pero continúa utilizándose "web".

Web services: Son servicios utilizados para transmitir y recibir datos por aplicaciones heterogéneas de diferentes empresas u organizaciones. Estos servicios son implementados en servidores Web, y la utilización de estos

servicios permitirá una integración total de la información, por lo cual se le considera una de las tecnologías más prometedoras en la actualidad.

WYSIWYG (What You See is, What You Get): Se utiliza para indicar que lo que se ve en pantalla, es como se vería de forma impresa.

- X -

XHTML: HTML extensible. Una reformulación de HTML como formato XML. Este lenguaje ejecuta las leyes de conformidad con XML, como las etiquetas de cierre obligatorias, y es normalmente el resultado que se obtiene cuando se aplica una hoja de estilo XSL a un archivo XML. La mayoría de los navegadores de Internet muestran XHTML como si fuera HTML normal, porque las diferencias son mínimas.

XML: Extensible Markup Language, o lenguaje de marcación extensible. Este formato toma prestados muchos elementos del lenguaje SGML pero está diseñado para fines distintos. Mientras que HTML es básicamente un lenguaje de presentación en el que intervienen elementos de formateado del contenido de texto como colores, tipos de letra, bordes, etc., el objetivo de XML es proporcionar un marco para el almacenamiento o intercambio de contenido rico en estructura.

XSL: Extensible Stylesheet Language, o lenguaje extensible de hojas de estilo. Formato XML que define el estilo y la información sobre la presentación que pueden aplicarse a un documento XML. El producto resultante puede ser cualquier tipo de documento, incluso un archivo sólo de texto si se desea, aunque habitualmente es un documento XHTML para ser visualizado a través de un navegador.

- Z -

ZIP: Término utilizado para describir la acción de comprimir uno o varios archivos con el fin de que ocupen menos espacio.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

B.1. Referencias bibliográficas y direcciones de Internet

[1] SPIRKIN, Alexander. Dialectical Materialism.

[2] MONK, Ray y RAPHAEL Frederic. The Great Philosophers - From Socrates to Turing.

[3] WIKIPEDIA. El método científico. http://es.wikipedia.org/wiki/M%C3%A9todo_cient%C3%ADfico

[4][5] PRESSMAN, Roger. Ingeniería del Software. Un enfoque práctico.

[6] BUZAN, Tony. Supercreativity.

[7] Jones, C. *Patterns of Software Systems Failure And Success.* International Thomson Publishing, Ene. 1996.

[8]MacCormak, A. Product-Development Practices That Work. *MIT Sloan Management Review.* Volume 42, Number 2.

[9]Steve McConnell is Chief Software Engineer at Construx Software. His first two books (*Code Complete* and *Rapid Development*) won *Software Development* magazine's Jolt Excellence award for outstanding software development books of their respective years. Steve has also written *Software Project Survival Guide*, *Professional Software Development*, and numerous technical articles. He is past Editor in Chief of *IEEE Software* magazine, and in 1998 readers of *Software Development* magazine named Steve one of the three most influential people in the software industry along with Bill Gates and Linus Torvalds. *Code Complete*, 2d Edition is now available.

[10] Jones, C. *Applied Software Measurement*, NY: McGraw-Hill

[11] Boehm, B. and Papaccio, P. 1988 Understanding and Controlling Software Costs. *IEEE Transactions and Software Engineering*, Oct 1998.

[12] La **Ley de Amdahl**, llamada así por el arquitecto de ordenadores Gene Amdahl, se usa para averiguar la mejora máxima de un sistema cuando solo una parte de éste es mejorado. http://es.wikipedia.org/wiki/Amdahl

[13] Bass, Len. Clements, Paul. Kazman, Rick. Software Architecture in Practice. Addison Wesley. 1998.

[14] Christine Hofmeister, Robert Nord, Dilip Soni, Applied Software Architecture. Addison-Wesley. 1st edition 1999

[15] Beck, Kent. Extreme Programming explained. Reading, Mass: Addison-Wesley Longman, Inc. 2000.

[16]http://www.sei.cmu.edu/publications/documents/97.reports/97tr029/97tr029title.htm

[17] http://www.sei.cmu.edu/risk/main.html

[18] M. Carr et all. Taxonomy based risk Identification. SEI 1993 http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.006.html

[19] Manifesto for Agile Software Development: http://www.agilemanifesto.org/

[20] Brian Gallagher. Taxonomy of operational risk. http://www.sei.cmu.edu/risk/taxonomy.pdf

[21] Software Engineering Institute (SEI). http://www.sei.cmu.edu/

[22] Emilio Rasic, "El rol de los Arquitectos de Software" http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=7

[23] García Martínez, R y Britos P. 2004. *Ingeniería de Sistemas Expertos.* Editorial Nueva Librería. ISBN 987-1104-15-4

[24] Steiner, G. A. 1999. *Planeación Estratégica.* Compañía Editorial Continental, S.A. de C.V. México.

[25] Sapag Chain, N. 1995. *Criterios de Evaluación de Proyectos.* McGraw-Hill.

[26] Glagovsky, H. E. 1996. *Esto es FODA.* Facultad de Ciencias Económicas de la Universidad de Buenos Aires. http://www.monografias.com/ Página web vigente al 03-07-05.

[27] Bueno, E. & Morcillo, P. 1993. *La Dirección Eficiente.* Ediciones Pirámide, S.A. Madrid. Segunda Edición.

[28] Piattini M., Calvo Manzano J., Cervera J y Fernández L. 1996. *Análisis y diseño detallado de Aplicaciones Informáticas de Gestión.* Editorial Ra-Ma.

[29] Presman R. 1993. *Ingeniería del software. Un enfoque práctico.* Editorial Mc Graw Hill. Tercera edición.

[30] TenStep Inc. 2002. *Proceso de Administración de Proyectos.* http://www.tenstep1.com.mx/

[31] Goldratt E. 1994. *La meta.* Tercera edición. Editorial Castillo. México

[32] Fernández Arena, J. A. 1986. *Elementos de Administración.* Editorial Diana. México

[33] Boehm, 1988: Boehm, B.: "A Spiral Model of Software Development and Enhancement". IEEE Computer. Vol. 21, # 5. 1988.

[34] DoD, 1994: DoD. Military Standard 498: Software Development and Documentation. Departament of Defense of the United States of America, 1994.

[35] Edwards, 1991: Edwards M., Howell S.: A methodology for system requirements specification and traceability for large real- time complex systems. Technical Report, Naval Surface Warfare Center, 1991.

[36] García, 2000: García Ávila, Lourdes. Modelo para la evaluación de la calidad del análisis y diseño orientados a objetos de sistemas informáticos (CADOOSI). Tesis de doctorado. Universidad Central de Las Villas, 2000.

[37] Jacobson, et al., 1998: Jacobson, I., Booch, G., Rumbaugh, J.: "The Unified Software Process". Adison- Wesley. 1998.

[38] Minasi, 2000: Minasi, Mark: The Software Conspiracy: Why Sftware Companies Put Out Faulty Products, How They Can Hurt You, and What You Can Do. McGraw-Hill, 2000.

[39] Palmer, 1997: Palmer J. D.: Traceability in Software Requirements Engineering, R.H. Thayer and M. Dorfman (Eds), IEEE Computer Society Press, 1997. (364:374).

[40] Sawyer y Kontoya, 1999: Sawyer, P. y Kontoya, G.: SWEBOK: "Software Requirements Engineering Knowledge Area Description". Informe Técnico Versión 0.5, SWEBOK Project, 1999. Disponible en http://www.swebok.org.

[41] Denger, C. Medina M. 2003. *Test Case Derived from Requirement Specifications.* Fraunhofer IESE Report.

[42] Binder R. V. 2000. *Testing Object-Oriented Systems.* Addison-Wesley. USA.

[43] Gutiérrez, J.J. Escalona M.J. et-al. 2006. Generation of test cases from functional requirements. A survey. 4º Workshop on System Testing and Validation. Germany.

[44] Gutiérrez J.J. Escalona M.J. et-al. 2005. A practical approach of Web System Testing. *Advances in Information Systems Development.* pp. 659-680. Sweeden.

[45] Ostrand T. J., Balcer M. J. 1988. Category-Partition Method. *Communications of the ACM.* 676-686.

[46] Object Management Group. 2002. *The UML 2.0 Testing Profile.* www.omg.org

[47] Offutt, J. et-al. 2003. Generating Test Data from Sate-based Specifications. *Software Testing, Verification and Reliability.* 13, 25-53. USA.

[48] DeMarco, Tom y Lister, Timothy. *Peopleware : Productive Projects and Teams.* Ed. Dorset House Publishing Company, Incorporated, 2a. edición.

[49] Walker, David. *Hard numbers on good work environments* y Spolsky, Joel. *Anyone managing software projects should read this!* Comentarios acerca del libro Peopleware en Amazon.com. http://www.amazon.com/Peopleware-Productive-Projects-Teams-Ed/dp/0932633439

B.2. Libros recomendados

1. El proceso unificado de desarrollo de software. Jacobson, I. – Booch, G. – Rumbaugh, J. Addison Wesley 2000 (ISBN 84-7829-036-2)

2. The Rational Unified Process. Ph. Kruchten. Addison Wesley 2000

3. Ingeniería de software. Un enfoque práctico. Pressman, R. Quinta edición. Mc. Graw Hill 2002 (ISBN 84-481-3214-9)

4. Ingeniería de software. Sommerville, I. Sexta edición. Addison Wesley 2002 (ISBN 970-26-0206-8)

5. Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Addison-Wesley, 1995.

6. Clay Carr: *Team Leader's Problem Solver.* Prentice-Hall, 1996

7. Craig Larman: *UML y Patrones: Introducción al análisis y diseño orientado a objetos y proceso unificado.* Prentice-Hall Hispanomericana, 2002. disponible a mediados de octubre 2002.

8. Cay Horstmann, Gary Cornell. *Core Java 2. Volume 1: Fundamentals.* Prentice-Hall, 1999.

9. Michael Blaha y William Premerlani: *Object-Oriented Modeling and Design for Database Applications.* Prentice-Hall, 1998.

10. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *A System of Patterns: Pattern-Oriented Software Architecture.* Wiley, 1996.

11. Luiz Fernando Capretz y Miriam A. M. Capretz: *Object-Oriented Software: Design and Maintenance.* Series on Software Engineering and Knowledge Engineering, vol. 6. World Scientific, 1996.

12. Martin Fowler: *Analysis Patterns: Reusable Object Models.* Addison-Wesley, 1997.

13. Martin Fowler: *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

14. Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

15. Ivar Jacobson, Grady Booch y James Rumbaugh: *The Unified Software Development Process.* Addison-Wesley, 1999.

16. John Vlissides: *Pattern Hatching: Design Patterns Applied.* Addison-Wesley, 1998.

17. Jos Warmer y Anneke Kleppe: *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1999.

18. Clay Carr: *Team Leader's Problem Solver.* Prentice-Hall, 1996.

19. S. Robbins: *Comportamiento Organizacional: Teoría y Práctica.* Octava edición. Prentice-Hall Hispanoamericana, 1999.

20. Watts Humphrey: *Introduction to the Personal Software Process (TM).* Addison-Wesley, 1997.

21. Capers Jones: *Assessment and Control of Software Risks.* Yourdon Press, 1994.

22. Roger S. Pressman. *Ingeniería del Software: Un Enfoque práctico.* Cuarta edición. McGraw-Hill, 1998.