Universidad de las Ciencias Informáticas

Facultad 3

Título: SISTEMA DE GESTIÓN DE LA INFORMACIÓN PARA EL DIAGNÓSTICO TECNOLÓGICO A ENTIDADES LEGALES CUBANAS

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autoras: Odalis Acosta Imbert
Aimé Rodríguez Martínez

Tutoras: Ing. Isabel Sánchez Pérez
Ing. Daylenis Ortíz Franco

Co-tutor: Ing. Darían González Ochoa

Junio del 2012

DECLARACIÓN DE AUTORÍA

Declaramos que somos las únicas autoras de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____.

_____ _____

Odalis Acosta Imbert Aimé Rodríguez Martínez

_____ _____

Ing. Isabel Sánchez Pérez Ing. Daylenis Ortíz Franco

AGRADECIMIENTOS

Es mi deseo expresar mi inmenso agradecimiento:

A mis mosqueteros por esa unión y fortaleza que me fue demostrada.
A mi esposo por su comprensión, por su apoyo incondicional, por darme tanta fuerza y sobre todo por su amor.
A mis tutoras, por su paciencia, dedicación, exigencia profesional; en fin, por ser parte de esto.
A mis amigos que son una parte importante en mi vida.
A todos mis profesores, que durante el transcurso de mi carrera fueron capaces de brindarme sus conocimientos.
A mis compañeros de estudio, por su cariño y amistad.
A todos aquellos que de una forma u otra hicieron posible este trabajo.
A mi compañera de tesis Aimé por su esfuerzo y perseverancia.
A la importante obra de nuestra Revolución.
A todos aquellos, que no menciono porque sería una lista considerable, pero que están en el mejor lugar que puedo reservarles, mi pensamiento.

Eternamente gracias.

Odalis Acosta Imbert

Quisiera agradecer primeramente al Comande en Jefe Fidel Castro Ruz por haber concebido la idea de esta universidad de ensueño que me ha abierto las puertas del conocimiento y que me ha hecho mejor persona.

A las personas que me han apoyado en la realización de este trabajo de diploma y en especial:

A mis padres que son mi fuente de inspiración, la de superarme cada día, sin su apoyo hubiese sido imposible llegar a convertirme en lo que soy.

A mis maestros y profesores que han inculcado en mí la perseverancia para poder lograr las metas más difíciles.

A todos mis compañeros en estos años de tristezas, alegrías, sacrificios, a los que una forma u otra han hecho posible este momento.

A mis tutoras que nos han apoyado y ayudado para que este trabajo esté terminado.

A mi compañera de tesis Odalis que a pesar de habernos conocido este curso no nos impidió realizar este trabajo con el cual terminan nuestras agonías.

Aimé Rodríguez Martínez

DEDICATORIA

A mi abuelita Gladis y a mis mosqueteros por ser mi apoyo, mi guía, mi fuerza y mi valor; especialmente a mi madre por confiar en mí y por enseñarme el valor de la fe y el amor.

Odalis Acosta Imbert

Quisiera dedicarle este trabajo de diploma a todos aquellos que de una forma u otra han estado conmigo:

A mi mamá y mi papá que son mi ejemplo a seguir y me siento orgullosa de ellos.

A mi hermana por ser mi mejor amiga, por estar ahí a mi lado cuando más la necesito. Gracias por tener fe en mí que al igual que tú yo lo podía lograr.

A mi tía Elo y a mi abuelita por criarme y quererme incondicionalmente, esto es para ustedes.

A mi novio, sin él no hubiese sido posible llegar hasta aquí, eres muy especial para mí, gracias por creer en mí cuando yo no lo hice.

A Rafa que me ha ayudado en la revisión de este trabajo.

A Yurisel que más que una amiga ha sido otra hermana para mí.

A mis amigos del pre Liliette, Silvio, Xenia, Jorge, Carlos, Amed y Willito.

A mis compañeros de estudio que compartimos buenos y malos momentos en la universidad: a los del grupo 3102, a Manuel que ha sido como un hermano para mí, a mis compañeros de CDI.

A mis amigos de Venezuela, que nunca olvidaré.

Aimé Rodríguez Martínez

RESUMEN

Se presenta un Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas. El objetivo del sistema es el de ayudar a manejar la información de manera segura e íntegra, para facilitar la toma de decisiones vinculadas a los procesos de informatización de estas instituciones y contribuir al proceso de despliegue de los proyectos del Centro de Gobierno Electrónico de la facultad 3 de la Universidad de las Ciencias Informáticas.

El sistema brinda la posibilidad de graficar la infraestructura de redes e informática de la entidad diagnosticada, con el objetivo de aportar el plano gráfico como parte de la información que facilitará las disposiciones posteriores. Se aporta un componente destinado al tratamiento de la seguridad del archivo de la entidad, donde se guardan los elementos capturados en la misma.

Para el desarrollo del sistema se utiliza la metodología ágil Scrum, además se hizo un estudio de los sistemas existentes, las herramientas y las diversas tecnologías referentes al desarrollo de software, de manera que se seleccionaron las adecuadas para la construcción del mismo. La propuesta está validada por el equipo de calidad de la facultad 3, con la utilización de las pruebas de caja negra, pruebas de caja blanca y la verificación de su diseño a través de métricas seleccionadas, obteniendo resultados satisfactorios que avalan el sistema aportado.

PALABRAS CLAVES

Sistema de Gestión de la Información, proceso de diagnóstico tecnológico.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad el desarrollo alcanzado por las tecnologías es indetenible, cada día aumentan más las necesidades y ofertas tecnológicas, con ello el conocimiento y desarrollo de la sociedad en general. Resulta inusual hoy en día, la existencia de empresas que no hagan uso de los avances tecnológicos.

En Cuba la introducción de las tecnologías en los diferentes sectores de la sociedad constituye una tarea fundamental que tiene como misión principal la de llevar al país a estar en correspondencia con los momentos actuales y a aumentar su desarrollo económico. El proceso de informatización en la industria cubana y sus diferentes empresas es uno de los principales ejemplos de desarrollo tecnológico en Cuba y a la vez es la principal meta para los trabajadores del Ministerio de la Informática y las Comunicaciones. Para llevar a cabo un correcto proceso de informatización en un departamento, empresa o industria es necesario haber desarrollado un buen diagnóstico que avale las condiciones existentes del local a informatizar.

El proceso de diagnóstico tecnológico es aquel que permite realizar un estudio en los locales donde se necesite desplegar un producto informático, con el objetivo de tomar las medidas necesarias sobre la información recopilada acerca de las condiciones y tecnologías que se requieren para instalar un software o aplicación informática. Este proceso es una colaboración conjunta entre la empresa y especialistas externos, que determinan las necesidades y el potencial tecnológico de la entidad (Carreras, 2009).

El proceso de diagnóstico tecnológico es el punto de partida del proceso de despliegue de un software en cualquier empresa. El resultado del mismo ofrece información detallada de todo lo que se tiene hasta el momento en cuanto a las condiciones requeridas para la implantación de un software, además se informa acerca de lo que se tiene y por consiguiente se deduce lo que hace falta.

Objetivos del diagnóstico tecnológico:

- Detectar el potencial tanto material como humano de la empresa para poder enfrentar el sistema a implantar.
- Analizar la dinámica interna de la empresa, para poder determinar el grado actual de adaptación a las nuevas tecnologías.
- Analizar el entorno en el que la entidad desarrolla su actividad.
- Cuantificar las inversiones necesarias para la adopción de las nuevas tecnologías.

Beneficios que proporciona el diagnóstico tecnológico:

- Ofrece una visión global de la empresa brindando así elementos valiosos que favorecen la toma de decisiones.
- Supone un estudio detallado del uso de las nuevas tecnologías en la empresa para poder obtener un mayor partido de ellas.
- Cuantifica las inversiones que son oportunas para la empresa.

El proceso de diagnóstico tecnológico previo al despliegue de los productos informáticos desarrollados en el Centro de Gobierno Electrónico (CEGEL) de la Facultad 3 de la Universidad de las Ciencias Informáticas (UCI) se realiza actualmente de forma manual en formato duro, provocando que la tarea se torne engorrosa y poco confortable. La gestión de esta información recopilada es fruto de un proceso extenso y en ocasiones agotador, debido a la manipulación de tantos datos para la confección del informe final y la gestión de la información para la toma de decisiones. La experiencia en diagnósticos anteriores arrojó que pudo haberse recogido mejor la información e incluso aumentar la cantidad de datos recopilados. Actualmente la captura de los mismos dificulta y ralentiza la obtención de resultados. El CEGEL no cuenta con un sistema informático que sea utilizado en el diagnóstico tecnológico.

Esta situación ha generado la necesidad de desarrollar una investigación que tiene como **problema a resolver**: ¿cómo contribuir a eliminar las deficiencias del proceso de diagnóstico tecnológico a entidades legales cubanas para facilitar la toma de decisiones en el proceso de despliegue?

Para darle solución a esta interrogante, se traza como **objetivo** de la investigación: desarrollar un Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas que contribuya a facilitar la toma de decisiones en el proceso de despliegue de los proyectos del CEGEL de la Facultad 3 de la UCI.

Para lograr el objetivo del presente trabajo se tiene como **objeto de estudio** el proceso de diagnóstico tecnológico, enfocado en el **campo de acción** Sistemas de Gestión de la Información para el proceso de diagnóstico tecnológico a entidades legales cubanas.

La **idea a defender** se formula a partir de que el desarrollo de un sistema de gestión de la información contribuirá a eliminar las deficiencias del proceso de diagnóstico tecnológico a

entidades legales cubanas para facilitar la toma de decisiones en el proceso de despliegue de los proyectos del CEGEL.

Como **tareas de la investigación** trazadas para dar solución al objetivo expuesto anteriormente, se tienen las siguientes:

- Estudio de los Sistemas de Gestión de la Información y los procesos de diagnóstico tecnológico.
- Estudio de las principales tecnologías y herramientas a utilizar para el desarrollo de la solución propuesta.
- Análisis y diseño de la funcionalidad referente a la recogida de la información a procesar en el proceso de diagnóstico.
- Análisis y diseño de la funcionalidad que permitirá graficar la infraestructura tecnológica de las entidades diagnosticadas.
- Implementación de la funcionalidad referente a la recogida de la información a procesar en el proceso de diagnóstico.
- Implementación de la funcionalidad que permitirá graficar la infraestructura tecnológica de las entidades diagnosticadas.
- Validación del sistema propuesto.

Métodos Teóricos

- Análisis-Síntesis: Se utilizó durante el proceso de revisión bibliográfica y particularmente para el estudio de los Sistemas de Gestión de la Información para el diagnóstico tecnológico y las principales tecnologías y herramientas utilizadas a nivel nacional e internacional.
- Modelación: Se utilizó en el diseño de los diferentes diagramas que forman el análisis y diseño del sistema.
- Sistémico: Sirvió para la descomposición del sistema en partes que luego se integró para formar parte de la herramienta Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas resultante como un todo.

Métodos Empíricos:

- Entrevista: Se utilizó para obtener información por parte del cliente acerca de los requisitos funcionales y no funcionales del sistema en cuestión.

- Observación: A través de la observación se identificaron aspectos importantes como son los requerimientos indispensables de este Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas.
- Medición: Se utilizó este método para la validación del sistema.

APORTE PRÁCTICO ESPERADO

Un Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas que facilitará la toma de decisiones en el proceso de despliegue de los proyectos del CEGEL.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción.

En este capítulo se definen los Sistemas de Gestión de Información y su vinculación a los procesos de diagnóstico tecnológico a entidades legales cubanas. Se incluyen todos los aspectos teóricos que soportan la aplicación, incluyendo el análisis comparativo de las metodologías de desarrollo de software, las tecnologías y las herramientas utilizadas en el proceso de desarrollo de este trabajo. Además se estudian los patrones arquitectónicos a utilizar, así como la seguridad en los sistemas informáticos, se fundamentará en cada caso el aspecto elegido.

1.2. Fundamentación del tema.

1.2.1. Proceso de diagnóstico tecnológico.

El diagnóstico es el resultado de la integración simultánea de múltiples datos procedentes de distintas fuentes informativas (técnicas, contexto, acciones y resultados) y recogidas con diferentes técnicas (Córdova Urbano, 2008).
El diagnóstico tecnológico es el punto de partida para la identificación de las necesidades (demandas tecnológicas) y las oportunidades (ofertas tecnológicas) en las empresas que desean informatizar sus servicios y funcionamiento en general (Carreras, 2009).

1.2.2. Sistemas de Gestión de la Información (SGI).

La gestión de la información comprende las actividades relacionadas con la obtención de la información adecuada, a un precio adecuado, en el tiempo y lugar adecuado, para tomar la decisión adecuada (Woodman, 1985).
Los SGI para la gestión del conocimiento constituyen hoy una alternativa de imprescindible presencia en cada organización, al permitir operar casi todos los activos tangibles e intangibles de la institución, convirtiéndose en la herramienta integral de gerencia más cotizada y necesaria para alcanzar con éxito los resultados propuestos por la organización.

En un SGI se manipula la información que se genera dentro y fuera de la organización. Este está orientado a la obtención de ventajas competitivas y a proveer a la organización de servicios de calidad, mediante el uso de tecnologías, en función de satisfacer las necesidades de información de sus clientes y alcanzar el desarrollo estratégico y las metas de la organización. La gestión por sí sola efectúa varias actividades, estas podrían ser: entrada, almacenamiento, procesamiento y salida de información. La entrada de datos es un proceso donde se recopila información para ser procesada en su mayoría o totalidad según la relevancia de esta, la cual puede realizarse de forma manual o automática. El almacenamiento es la actividad que permite reservar la información para ser usada en otro momento, asegurando la disponibilidad de la misma. El procesamiento de información está orientado a la capacidad que posee el SGI para gestionar el flujo de información de acuerdo a restricciones y operaciones establecidas por el lenguaje de programación. Por último, el proceso de salida posibilita utilizar la información previamente entrada o almacenada, ya sea mostrándola en una interfaz al usuario o sirviendo de entrada a otros sistemas internos o externos (Cumming, 2001).

1.2.3. Sistema de Gestión de la Información para la toma de decisiones en un proceso de diagnóstico.

El SGI contribuye a la toma de cualquier decisión basada en información, ya que una vez que está siendo procesada con este sistema, resulta más fácil tomar una decisión, considerándola como el conjunto de acciones adoptadas en un momento específico, resultado de la aplicación de ciertas reglas y políticas a las condiciones particulares existentes en el momento.
Se puede comprender de esta manera, cómo un SGI puede ayudar a realizar un proceso de diagnóstico tecnológico, pues en este último es necesario agrupar, organizar y clasificar la información.

1.2.4. Sistemas de Gestión de la Información en el mundo.

En el mundo existen diferentes SGI de acuerdo a su utilidad final y las organizaciones para las que están destinados. Entre ellos se cuentan:

Proyecto SIRED (Sistema de Información de Redes)

La solución SIRED, diseñada conjuntamente por la Unidad Funcional de Sistemas de la Información y por la Unidad de Estructura Territorial, integra la gestión y mecanización de cualquier operación o actividad que deban realizar las entidades de la red en cualquier punto de España, procedente de los diversos negocios de seguro directo. El sistema dispone de varios “perfiles de acceso” según la naturaleza del trabajo que deba desarrollar cada operador , así como herramientas de ayuda, tales como un sistema de calendario- agenda (para usos comerciales y profesionales) y un software de gestión documental para la ayuda en la búsqueda de documentos, pólizas , recibos, expedientes (Martínez Martínez, 2004).

INFOTEC (Información Tecnológica)

Es un Sistema Regional de la Información Científica, Tecnológica y de Innovación del sector agropecuario. Se propone como un componente de la solución sectorial, altamente flexible y con capacidad de adaptarse a las diferentes etapas de transición en la adopción institucional e individual de las nuevas tecnologías de información y comunicaciones. Facilita a los usuarios finales el acceso a la información científica, tecnológica y de innovación en el sector agropecuario y de desarrollo rural. Además dinamiza el flujo del conocimiento y la información entre los responsables de la investigación y desarrollo agropecuarios en la región, en un esfuerzo orientado a fortalecer y consolidar el Mercado Regional de Tecnologías Agropecuarias. La característica particular que le da identidad a INFOTEC es su potencial para identificar y articular las fortalezas específicas de los diversos actores, en un sistema altamente sinérgico que incrementa la eficiencia de los recursos humanos, tecnológicos, de infraestructura, administrativos, de políticas, de información y económicos, aplicados a la investigación y el desarrollo tecnológico del sector agropecuario, incrementando su competitividad y potencial de sustentabilidad (IICA., 2000).

1.2.5. Sistemas de Gestión de la Información en Cuba

Cuba no está al margen del desarrollo de estos sistemas, muestra de ello es el Sistema Automatizado para el Diagnóstico de la Gestión de la Información y Conocimiento en la Empresa, basado en la propuesta del Modelo operacional de gestión de información y conocimiento para la empresa cubana en perfeccionamiento (MOGICEP) (Artiles Visbal, 2006).

También cuenta Cuba con una herramienta para el diagnóstico de la gestión de información en la empresa, mediante proyectos de consultoría de Gestión del Conocimiento y la Tecnología (GECYT) (GECYT, 2011).

Son todos ejemplos de la utilidad de los SGI para el desarrollo de la economía cubana. Sin embargo, ninguno de estos sistemas satisface las necesidades actuales que llevan a la realización de este trabajo. Cada una de estas aplicaciones tiene especificaciones de acuerdo a la entidad donde se utiliza, lo que no las hace reutilizable para otras funciones, como es el caso de las que se manejan en el CEGEL de la facultad 3 de la UCI.

1.3. Metodologías de desarrollo

Se puede decir que, una metodología de desarrollo de software es un proceso que guía a los desarrolladores a los que les brinda métodos y herramientas, proporcionándoles una ayuda muy importante e indispensable para que el producto final posea las funcionalidades requeridas por el cliente y que cumpla con las necesidades del mismo y del usuario final, es una secuencia de actividades organizadas y bien pensadas que transforman los requisitos del cliente en el producto final (Carrillo Pérez, y otros, 2008).

Para el desarrollo del sistema, se necesita una metodología liviana, configurable, adaptable, de poca documentación aunque no nula, que responda a los intereses del cliente y el producto, que mantenga al cliente interesado, que le muestre productos funcionales en poco tiempo, y que permita que estos productos sean simples y fáciles de modificar. Existen diversas metodologías ágiles, que por su potencial son utilizadas comúnmente a nivel mundial. A pesar de esto, no serán objeto de atención en el presente estudio, debido principalmente al tamaño del equipo de desarrollo que requieren para una mayor eficiencia. Es el caso de la metodología Cristal, la cual necesita como mínimo tres integrantes. Esto no coincide con el equipo en cuestión, pues se cuenta con solo dos integrantes. También influyen los documentos que generan, como es el caso del Proceso Unificado Ágil (AUP), el cual es el Proceso Unificado Racional (RUP) ágil, a pesar de fusionar sus flujos para lograr agilidad en su desempeño, en cuanto a los artefactos es bastante radical, pues para eliminar algunos de los que plantean en su matriz (RUP) es necesario llevar una serie de actividades y cumplir algunos requisitos que deriven el uso o no de dichos artefactos, aspecto que afectaría el tiempo de desarrollo de la aplicación.

Por estas razones, se centrará el estudio comparativo en la Programación Extrema (XP) y Scrum, ambas metodologías ágiles, seleccionadas por ser de las más utilizadas en la universidad y que han aportado resultados satisfactorios, además por ser las más cercanas a los intereses que se persiguen con el desarrollo del sistema.

1.3.1. Extreme Programming (XP)

Desarrollada por Kent Beck, XP es la primera metodología ágil y la que le proporcionó conciencia al movimiento actual de metodologías ágiles. De la mano de Kent Beck, XP ha conformado un extenso grupo de seguidores en todo el mundo, produciendo una gran cantidad de libros a los que dio comienzo el mismo Beck (Beck, 2000). El diseño de XP propone restar importancia al análisis como fase independiente, puesto que se trabaja exclusivamente en función de las necesidades que van surgiendo. Con esta metodología se logra incluir al cliente como parte del equipo de desarrollo, lo que aportará mayor valor de negocio y facilitará que los programadores puedan resolver de manera inmediata cualquier duda asociada, se recomienda para esto hacer la selección de un representante que actúe como interlocutor.
El ciclo de vida de un proyecto XP consta de seis fases:

- Exploración: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.
- Planificación de la entrega: En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Esta fase dura unos pocos días.
- Iteraciones: Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior.
- Producción: La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo

tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

- Mantenimiento: La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.
- Muerte del proyecto: Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo. XP propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito (Beck, 2000).

1.3.2. Scrum

"Scrum" es una metodología para la gestión de proyectos, expuesta por HirotakaTakeuchi e Ikujiro Nonaka, en el artículo *The New New Product Development Game*. "El precepto básico de esta metodología es que el mercado competitivo de los productos tecnológicos, además de los conceptos básicos de calidad, coste y diferenciación, exige también rapidez y flexibilidad". (Takeuchi, 1999).

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicada para proyectos donde se necesita obtener resultados con prontitud, donde los requisitos son cambiantes o poco definidos, donde la innovación, la flexibilidad y la productividad son fundamentales.

Características de Scrum:

- Es una metodología de desarrollo ágil.
- Está pensada para equipos de desarrollos pequeños (no más de 8 personas).
- Se especifican pocos artefactos eliminando el "papeleo" innecesario y dedicando más tiempo a la implementación.
- Permite la entrega de un producto funcional al finalizar cada ciclo.
- Da la posibilidad de ajustar la funcionalidad en base a la necesidad de negocio del cliente.

- Permite hacer una visualización del proyecto diaria.
- Tiene un alcance acotado y viable.
- Se aplica en equipos integrados y comprometidos con el proyecto y que se auto administran.
- No es una metodología de análisis, ni de diseño (aunque puede adaptarse para que lo sea) sino de gestión del trabajo.
- Puede ser aplicada a distintos modelos de calidad (como podría ser el *Capability Maturity Model Integration* (CMMI) por sus siglas en inglés) puesto que estos dicen qué hacer, es decir, dicen que hay que gestionar el proyecto, pero no dicen cómo. Ahí es donde entra Scrum como modelo de gestión del proyecto.

No es ni la mejor metodología ni la única pero sí es la que está empujando muy fuerte por la facilidad de implantación y por su agilidad en cuanto a cambios y lo que propiamente aporta en comparación con otras. Por otro lado, Scrum no exige documentar nada para iniciar un proyecto, algo que en otras metodologías es indispensable. La idea principal es la de ponerse a trabajar prácticamente desde el primer momento y empezar a sacar frutos de ese trabajo para que el cliente vaya viendo los avances y se quede satisfecho con lo que se está haciendo y cómo se está haciendo. Hay dos aspectos fundamentales a diferenciar, los actores y las acciones. Los actores de forma general, serán:

- *Product Owner* (Propietario del Producto): conoce y marca las prioridades del proyecto o producto.
- *Scrum Master* (Maestro del Scrum): es la persona que asegura el seguimiento de la metodología guiando las reuniones y ayudando al equipo ante cualquier problema que pueda aparecer. Su responsabilidad es entre otras, la de hacer de paraguas ante las presiones externas.
- *Scrum Team* (Equipo de Scrum): son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el *Product Owner*.
- Usuarios o Clientes: son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden aportar ideas, sugerencias o necesidades.

Las acciones de Scrum forman parte de un ciclo iterativo repetitivo, por lo que el mecanismo y forma de trabajar que a continuación se indica, tiene como objetivo minimizar el esfuerzo y maximizar el rendimiento en el desarrollo.

Las acciones fundamentales de Scrum son:

- *Product Backlog* (Pila de tareas del producto): corresponde con todas las tareas, funcionalidades o requerimientos a realizar.
- *Sprint Backlog* (Pila de tareas del *Sprint* o Ciclo): corresponde con una o más tareas que provienen del *Product Backlog* de donde se saca una o más tareas que van a formar parte del *Sprint Backlog*. Una norma fundamental es que mientras un *Sprint Backlog* se inicia, este no puede ser alterado o modificado.
- *Sprint Planning Meeting* (Reunión de Planificación del Ciclo): es una reunión que tiene por objetivo, planificar el Sprint a partir del *Product Backlog*. El objetivo de esta reunión es la de mover las tareas del *Product Backlog* al *Sprint Backlog*. De esta reunión surge el *Sprint Goal*, que es un pequeño documento o una breve descripción que indica lo que el Sprint intentará alcanzar.
- *Daily Scrum Meeting* (Reunión Diaria de Scrum): es una tarea iterativa que se realiza todos los días que dure el *Sprint Backlog* con el equipo de desarrollo o de trabajo. Se trata de una reunión operativa, informal y ágil, de un máximo de treinta minutos.

El *Scrum Master*, debe eliminar aquí cualquier obstáculo que encuentre.

"Cuando se ha finalizado un *Sprint Backlog*, se debe tener un entregable que se pueda mostrar y que evidencie los avances acometidos en el Sprint."(Takeuchi, 1999).Todas las acciones de Scrum se muestran en la figura 1.

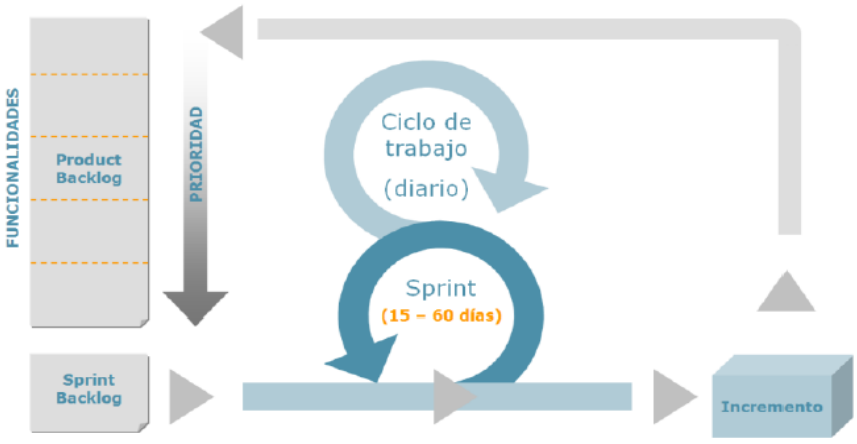

Figura 1. Scrum

Una vez terminado el Sprint si en el *Product Backlog* quedan tareas se planifica un descanso para el *Scrum Team* y se resuelven posibles cuestiones vistas respecto a los aspectos positivos (para repetirlos) y los aspectos negativos (para evitar que se repitan) del Sprint.

Comienza así una nueva fase en la que se comenzará nuevamente planificando un *Sprint Planning Meeting*.

Se escogió esta metodología debido a su facilidad de uso, teniendo en cuenta que el equipo de desarrollo tenía experiencia en el trabajo con ella. El sistema fue solicitado con premura por el cliente y esta es adecuada para proyectos en los que el tiempo de entrega es de vital cumplimiento. Se necesitaba además adaptar los artefactos generados en el proceso de desarrollo del software a los requerimientos inestables del cliente, y esta metodología muestra gran flexibilidad al respecto. En este punto XP y Scrum están bastante parejas, pero la primera tiene la limitante de que se recomienda tener al cliente como parte del equipo de desarrollo, no siendo posible en este caso, por tanto a pesar de las ventajas proporcionadas por XP, Scrum es la seleccionada para guiar el desarrollo del sistema en cuestión.
Una vez seleccionada la metodología a utilizar es necesario para modelar un proceso de desarrollo del software, utilizar el lenguaje de modelado apropiado, a continuación se explica cuál fue el escogido.

1.4. Lenguaje de modelado.

El desarrollo de software es un proceso que consta de diferentes actividades, entre las que se destacan: el modelado del negocio, el desarrollo de requisitos, el diseño arquitectónico, el diseño detallado, la programación y las pruebas de la aplicación (Sommerville, 2002). En cada una de estas actividades, el ingeniero de software debe elaborar diferentes tipos de modelos. Según las características del modelo que se persigue se utilizará el Lenguaje Unificado de Modelado (UML por sus siglas en inglés) como lenguaje representativo porque permite la representación gráfica de un sistema con tecnología orientada a objetos (Larman, 2004), los diagramas que se van a utilizar para documentar el sistema serán los diagramas de clases de diseño, aprovechando para estos los beneficios que aporta UML.

1.4.1. UML

Como su nombre lo indica, UML es un lenguaje de modelado visual que permite modelar la complejidad de los sistemas a analizar o diseñar, se centra en la representación gráfica de un sistema. Permite modelar sistemas utilizando tecnologías Orientadas a Objetos (OO) (Booch, et al., 1999).Las funciones fundamentales de UML son:

- **Visualizar**: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- **Especificar**: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir**: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar**: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

UML es una consolidación de muchas de las notaciones y conceptos más usados orientados a objetos. Empezó como una consolidación del trabajo de Grady Booch, James Rumbaugh, e Ivar Jacobson, creadores de tres de las metodologías orientadas a objetos más populares (D´Souza, y otros, 1998).

Se utilizará UML versión 2.0 para el desarrollo de los artefactos del presente trabajo. Escogido el lenguaje de modelado se analizarán las herramientas más utilizadas que soportan UML para seleccionar de ellas la más apropiada.

1.5. Herramientas de modelado

Las Herramientas CASE (*Computer Aided Software Engineering*) en español, Ingeniería de Software Asistida por Ordenador, son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática y documentación o detección de errores.

En el presente trabajo se han tomado en cuenta dos de las más usadas internacionalmente, además cumplen con las cualidades requeridas. Estas son Rational Rose y Visual Paradigm.

1.5.1. Rational Rose

El Rational Rose es una herramienta CASE desarrollada por Rational Corporation, basada en UML. Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema (lógico y físico). Permite crear y refinar vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software. Permite crear los diferentes diagramas que se generan en el proceso de ingeniería durante el desarrollo del software. Presenta un gran número de estereotipos que permiten el proceso de modelación del software. Esta herramienta es capaz de generar el código fuente de las clases definidas en el flujo de trabajo de diseño. Proporciona mecanismos para realizar ingeniería directa e inversa, posibilita la construcción de un modelo de casos de usos, identifica los objetos y representa cómo interactúan con los diagramas de secuencia y colaboración, así como otras operaciones. Además Rational Rose organiza sus diagramas en vistas: la vista de casos de uso, la vista lógica, la vista de componentes y la vista de despliegue. El uso de estas vistas facilita la organización del trabajo para una mejor comprensión del mismo. (Cámara, 2005).
Esta herramienta no es gratuita, por lo que se debe hacer un previo pago para poder adquirir el producto.

1.5.2. Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software. Permite generar código inverso, generar código desde diagramas y documentación. Es muy popular por sus disímiles características, producto de calidad, que soporta aplicaciones web, es muy fácil de instalar y actualizar. Permite la generación de código para varios lenguajes. Su diseño está centrado en casos de uso, enfocado al negocio generando un software de mayor calidad. Presenta capacidades de ingeniería directa e inversa y disponibilidad en múltiples plataformas.

Ventajas del Visual Paradigm:

- Ofrece entorno de creación de diagramas para UML 2.0.
- Disponibilidad en múltiples plataformas.
- Disponibilidad de integrarse en los principales IDE.
- Soporta una amplia gama de lenguajes en la generación de código e ingeniería inversa en Java, C++, CORBA IDL, PHP, Esquema de XML, Ada y Python.
- La generación de código soporta soporta C #, VB .NET, Lenguaje de Definición de Objeto (ODL), Flash Action Script, Delphi, Perl, Objetivo-C, y Ruby.
- Se integra con las herramientas Java (Eclipse/IBM WebSphere, JBuilder, NetBeans IDE, Oracle JDeveloper, BEA Weblogic).

Visual Paradigm for UML (VP-UML) es una herramienta potente, multiplataforma, fácil de usar, permite modelado con UML y ofrece a los desarrolladores de software de los elementos necesarios para construir aplicaciones de calidad más rápido (Paradigm, 2012).

Es una de las herramientas de modelado que está probada a nivel mundial, comúnmente usada, robusta, íntegra, de fácil aprendizaje, con abundante documentación y que soporta a UML como lenguaje de modelado. Para el modelado del sistema se escogió como herramienta CASE a Visual Paradigm en su versión 8.0, puesto que es una herramienta potente, con licencia que adquirió la UCI, además de tenerla como política de herramienta CASE a utilizar.

Luego de seleccionada la herramienta, es necesario escoger el lenguaje de programación a utilizar para la implementación del sistema. Para el desarrollo del mismo se optó por una aplicación de escritorio porque se desconocen las condiciones de la entidad a diagnosticar en cuanto a su conectividad, esto influye a la hora de guardar la información en la base de datos central por lo tanto se aprovecha de las ventajas que poseen. Pueden funcionar en todo tipo de entornos con conexión, sin conexión o con conectividad ocasional. Este tipo de aplicaciones puede ejecutarse solo en una máquina cliente o conectarse con servicios ofrecidos por un servidor. Las aplicaciones de escritorio ofrecen la mayor flexibilidad en cuanto a recursos, topologías y tecnologías a utilizar (de la Torre Llorente, y otros, 2010). A continuación se definirán los lenguajes estudiados para desarrollar la aplicación de escritorio.

1.6. Lenguaje de programación para aplicaciones de escritorio

Dependiendo del tipo de programa que se desee realizar hay diferentes lenguajes en los que se puede programar: Java, C++, Delphi, Visual Basic, Visual C# .NET, entre otros. Es importante tener en cuenta sus características para elegir el correcto. El estudio comparativo del lenguaje de programación estuvo condicionado por el conocimiento del equipo de desarrollo, teniendo en cuenta la premura del cliente referente al uso del sistema. Es por esto que a pesar de existir diversos lenguajes, con características importantes, se centra el estudio en Java y C#, con el objetivo de ganar en tiempo, aprovechar la facilidad de aprendizaje y las novedades que aportan.

1.6.1. C Sharp

CSharp (C#) es un lenguaje de programación orientado a objetos (object-oriented), de propósito general (general-purpose) y con seguridad en la definición de tipos (type-safe).Con este fin el lenguaje balancea la simplicidad, la expresividad y el rendimiento. El lenguaje C# es uniplataforma y fue escrito para trabajar con el *framework de Microsoft .NET.*

C# es una rica implementación del paradigma orientado a objetos que incluye la encapsulación, la herencia y el polimorfismo. Las características que distinguen a C# de la perspectiva orientada a objetos son:

- Sistema de tipo unificado

  El bloque fundamental de C# es una unidad encapsulada de datos y funciones llamadas type. C# es un sistema de tipo unificado (*unified type system*) donde todos los tipos comparten una base común, esto quiere decir que son objetos de negocio o tipos primitivos como números, que comparten las mismas funcionalidades básicas. Por ejemplo: cualquier tipo puede ser convertido a una cadena (*string*) llamando al método *ToString*.

- Interfaces y clases

  En el paradigma puro orientado a objetos el único tipo es la clase. En C# existen otros tipos tales como la interfaz, similar a la de Java. Una interfaz es parecida a la clase, excepto que es solo una definición por tipo, no una implementación. Es particularmente

útil en escenarios donde la multi-herencia es requerida, aspecto que no existe en lenguajes como C++ y Eiffel.

- Propiedades, métodos y eventos
  En el paradigma puro orientado a objetos todas las funciones son métodos. En C# los métodos son solo un tipo de función, que incluye propiedades y eventos. Las propiedades son funciones que encapsulan una pieza del estado del objeto, por ejemplo: el color de un botón. Los eventos son funciones que simplifican las acciones sobre los cambios de estado de los objetos.

C# es llamado un lenguaje fuertemente tipado porque sus reglas son muy estrictas, por ejemplo: un método al que se le pase por parámetro un entero no se le debe pasar una cadena *(Albahari, y otros, 2010)*.

1.6.2. Java

Java es un lenguaje de programación creado por SUN Microsystems muy parecido al estilo de programación del lenguaje “C” y basado en lo que se llama programación orientada a objetos (POO) (Muñoz, 2006).

Entre las principales características de Java se pueden citar:

- Simple: Java elimina aspectos confusos de C++ y muy poco utilizados como la sobrecarga de operadores.
- Orientado a objetos: la POO es un modo de desarrollar software describiendo problemas mediante el uso de elementos u objetos desde el espacio del problema y no mediante un conjunto de pasos secuenciales que se ejecutarán (Zukowski, 2007).
- Distribuido: la visión de Sun ha sido “la red es el ordenador”, con Java querían un lenguaje de programación que reconociera la red y que admitiera el acceso a objetos distribuidos tan fácilmente como a los objetos locales.
- Interpretado: el código Java es traducido en códigos de bytes no asociados a una plataforma.
- Seguro: El verificador de bytes proporciona el primer nivel de defensas. En segundo lugar está el cargador de clases que es el responsable de cargar las clases, este

permite que las clases cargadas desde diferentes posiciones de la red se mantengan separadas. Esto garantiza que una clase que no es del sistema nunca se confundirá con una clase del sistema.

Terence Parr, cofundador y jefe de investigación de jGuru.com (Zukowski, 2007) mencionaba que en este lenguaje, todos los códigos se ejecutan en una zona de pruebas con lo que se evita que el código malintencionado pueda ejecutar métodos privilegiados, saturando los búferes o cualquier otra cosa que pudiera poner en peligro la integridad del sistema. Para realizar una aplicación que no dependa de la plataforma o sistema operativo, siempre se piensa en Java primeramente, pues Java es multiplataforma, puede correr en Unix, Windows y otros sistemas con solo instalar la máquina virtual.

Aunque C# tiene muchas características que lo hacen deseable como la aparición de Mono para utilizarlo en el sistema operativo Linux, es un lenguaje cuya plataforma por excelencia es privativa por tanto, para desarrollar el sistema se selecciona Java por su independencia de plataforma, de esta forma se adentra el sistema en las políticas de migración con respecto al software libre que lleva a cabo la UCI.

Bajo estas fundamentaciones se decide utilizar este lenguaje para desarrollar este sistema. Una vez seleccionado el lenguaje de programación a utilizar se analizarán las herramientas que permitirán desarrollar el sistema, seleccionando la más apropiada.

1.7. IDE para Java.

El estudio comparativo de los IDE para Java estuvo condicionado, entre otras cosas, por el conocimiento del equipo de desarrollo, factor que influye en la entrega a tiempo del producto. La elección de un IDE para trabajar con Java es una decisión muy importante. El hecho de que Sun ponga a disposición gratuita el JDK de Java y otras descargas, ha permitido aflorar un cúmulo de aplicaciones de apoyo a los desarrolladores Java, incluyendo el IDE.

Existen y se han desarrollado a lo largo del tiempo muchos IDE como AnyJ, NetBeans, Sun Java Studio Creator, Borland, JBuilder, IBM y Eclipse.

La mayoría de estos IDE son propiedad de compañías como Sun, Borland o IBM. En el presente trabajo se han tomado en cuenta dos de los IDE más usados mundialmente, pues se

estudian y emplean en la universidad, además cumplen con las cualidades requeridas, Eclipse y Netbeans ambas para código abierto y multiplataforma.

1.7.1. Eclipse

Este es un IDE de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para Visual Age. Este IDE es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse es un IDE, puesto que provee herramientas para administrar áreas de trabajo (o *workspaces* en inglés), construir, lanzar y depurar aplicaciones, así como compartir artefactos con un equipo y versionar el código fuente. Es abierto, debido a que su diseño le permite ser extendido fácilmente por terceras partes. Ha sido utilizado exitosamente para construir ambientes para un amplio rango de temas como el desarrollo en lenguaje de programación Java, Servicios Web, certámenes de programación de juegos, entre otros (Arthorne, y otros, 2004).

1.7.2. NetBeans

NetBeans es un IDE libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans es un proyecto de gran éxito con una gran base de usuarios, y una comunidad en constante crecimiento. Sun Micro Systems fundó el proyecto de código abierto.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las API (por sus siglas en inglés, *Application Programming Interface*) de NetBeans y un archivo especial que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las

aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

La plataforma Netbeans es una base para desarrollar aplicaciones de escritorio complejas con un enfoque modular y pensando en características como la extensibilidad y la escalabilidad. El IDE Netbeans es una muestra del tipo de aplicaciones que se pueden desarrollar utilizando la plataforma, ya que el mismo está construido sobre ella. El uso de la plataforma permite un mejor aprovechamiento del tiempo de desarrollo, una mejor organización de la aplicación basada en estándares y patrones estructurales y de diseño, una arquitectura consistente y robusta, y un mejor rendimiento en cuanto a tiempo de ejecución y optimización de recursos (NetBeans, 2012).

NetBeans es el ambiente de desarrollo a utilizar. Aunque ambos, Eclipse y NetBeans, ofrecen similares recursos para el desarrollo de aplicaciones de escritorio, como es el caso que se presenta, NetBeans es la mejor opción debido a que nos proporciona otras características muy útiles como: gestor de ventanas, API de acciones, API para la creación de diálogos y *wizards*, así como la integración con *java help* entre otras muchas. Un IDE más que interesante si queremos estar preparados para el desarrollo de aplicaciones de escritorio en java. Además de ser la herramienta pedida por el cliente y de la cual tiene conocimiento el equipo de desarrollo.

Para el desarrollo del sistema fue imprescindible el estudio de los patrones arquitectónicos, los cuales permiten ahorrar tiempo en la construcción del software. Se explicará a continuación el patrón arquitectónico escogido para la implementación del sistema.

1.8. Patrones arquitectónicos.

Los patrones arquitectónicos expresan esquemas de organización estructural fundamentales para los sistemas de software. Estos se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el proceso, el refinamiento y el código.

Un patrón de arquitectura de software describe un problema en particular del diseño, en un contexto específico. Se define qué patrón arquitectónico usar en la fase inicial de desarrollo de software. Existen diversos tipos de patrones arquitectónicos, entre ellos:

- **N - Capas:** Ayuda a estructurar aplicaciones que pueden ser descompuestas internamente en grupos con distintos niveles de abstracción (granularidad).
- **Pizarrón:** Útil en problemas para los cuales no hay una solución conocida. Generalmente provee aproximaciones a la solución final.
- **Modelo - Vista - Controlador:** divide a una aplicación interactiva en tres componentes: modelo, vistas y controladores. MVC, son las siglas de modelo-vista-controlador (o en inglés, model-view-controller.
- **Presentación – Abstracción - Control:** define al sistema como una jerarquía de agentes que cooperan entre sí para implementar la funcionalidad de la aplicación.

Buschmann propone los patrones arquitectónicos como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. Expresan el esquema de organización estructural fundamental para sistemas de software. Estos proveen un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación (con amplitud de todo el sistema) y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software (Buschmann, y otros, 1996). Para la arquitectura del sistema se propone utilizar el patrón N-Capas, para aprovechar las ventajas que ofrece en cuanto al bajo acoplamiento entre las capas y la organización de las clases del sistema.

Después de haber estudiado algunos de los patrones arquitectónicos, es necesario para el desarrollo del sistema utilizar un gestor de base de datos que ayude a administrar la información, se analizarán a continuación los gestores de datos seleccionados por el equipo de desarrollo para su posterior uso.

1.9. Gestores de bases de datos

Un Sistema Gestor de Bases de Datos (SGBD) o en inglés *Data Base Management System* (DBMA) como su nombre lo indica, facilita la gestión de las bases de datos, en otras palabras, es una aplicación que actúa como mediador en forma de interfaz entre el usuario y la base de

datos facilitando la creación, uso y administración en general de la base de datos. Los SGBD tienen varias ventajas como son:

- Administración centralizada de datos.
- Integridad del sistema.
- Reutilización de datos.
- Control sobre la redundancia y consistencia de datos.
- Asociación y compartición de datos.
- Establecimiento de prioridades de requerimientos (Bertino, 1995).

El sistema a desarrollar necesita un gestor potente, con gran capacidad de almacenamiento, alto rendimiento, además de independencia tecnológica, seguridad y consistencia en los datos. Existen varios gestores que cumplen con estos requisitos, pero no serán objeto de estudio del presente trabajo por tener algunas importantes limitantes, es el caso de Oracle, uno de los más reconocidos mundialmente, posee soporte de transacciones, gran escalabilidad, estabilidad y es multiplataforma, sin embargo es propietario, lo mismo pasa con SQL Server, ambos limitarían la independencia tecnológica que se persigue.

El éxito del trabajo con base de datos radica principalmente en mantener la seguridad y la integridad de los datos de la misma. Es por eso que al escoger entre los sistemas gestores de base de datos, los que más resaltan por sus potencialidades, ventajas y teniendo en cuenta las necesidades de la aplicación son PostgreSQL y MySQL. A continuación se explicará cada uno para establecer sus diferencias.

1.9.1. MySQL

MySQL es un SGBD bajo la Licencia Pública General de GNU (GPL). Es actualmente uno de los servidores de base de datos más populares, con una estimación de más de diez millones de instalaciones. Es un pequeño y compacto servidor de base de datos, ideal para pequeñas y medianas aplicaciones. MySQL soporta el estándar SQL (ANSI), y además está disponible para distintas plataformas. Es relativamente sencillo de manejar y consume pocos recursos. Este gestor admite la comprobación de clave externa en las tablas *InnoDB* y las eliminaciones en cascada (Gilfillan, 2003).

Las principales características de este gestor de bases de datos son las siguientes:

- Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
- Soporta gran cantidad de tipos de datos para las columnas.
- Gran portabilidad entre sistemas.
- Soporta hasta 32 índices por tabla.
- Gestión de usuarios y contraseñas, manteniendo un muy buen nivel de seguridad en los datos.

La última versión hasta ahora de MySQL es la 5.6.

1.9.2. PostgreSQL

Es un gestor de base de datos relacional orientado a objetos, debido a que incluye características como la herencia, tipos de datos, entre otras. Es de código abierto, bajo licencia BSD la cual permite generar versiones comerciales del producto, soporta grandes volúmenes de datos y es altamente configurable, multiplataforma y su distribución es gratis. PostgreSQL incluye la implementación del estándar SQL92/SQL99 además corre en la mayoría de los sistemas operativos: Linux, varias versiones de UNIX, y Windows.

Este SGBD tiene la ventaja de que un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. Es considerado actualmente como uno de los sistemas gestores de base de datos objeto-relacional de código abierto más avanzado. También debe tomarse en cuenta la siguiente idea del fabricante: “Lo mejor de todo es que el código fuente PostgreSQL está disponible bajo la licencia open source: la licencia PostgreSQL”. Esta licencia permite la libertad para usar, modificar y distribuir PostgreSQL de cualquier manera que se desee, código abierto o propietario. Cualquier modificación, mejora o cambios que realice son propios para hacerlo como se desee (PostgreSQL, 2012).

Por ser uno de los gestores más usados en la universidad, por las características antes mencionadas y por el conocimiento del equipo de desarrollo, se decide optar por PostgreSQL 9.1 como SGBD a utilizar en este proyecto.

1.10. Base de datos portable

Como parte de la solución propuesta es necesario realizar un estudio de las bases de datos portables que garantizarán almacenar los datos, debido a que el uso de la aplicación tendrá dos momentos: uno sin conexión y otro conectado a un servidor, por lo tanto se hace imprescindible trasladar la información recogida de la entidad en un fichero. Existen varias bases de datos portables entre estas Berkeley y SQLite, de estas la más utilizada es SQLite, que tiene una comunidad encargada de mejorar y dar soporte a la misma, además esta base de datos portable es usada por grandes empresas como son:

- McAffe: Utiliza SQLite en el conocido programa antivirus de esta empresa.
- Google: Utiliza SQLite en productos como Desktop Mac y en el Google Gears.
- Adobe: Utiliza SQLite en su Photoshop Lightroom y próximamente en Air Proyect (SQLite, 2012).

Otro elemento que decidió el uso de SQLite es que en Berkeley las consultas se hacen por medio de *XPath* o *XQuery*, además los datos no se guardan en las clásicas tablas, si no que se hace en archivos XML (por sus siglas en inglés, *Extensible Markup Language*) con su estructura completa (BD, 2010), lo que conlleva al análisis de XML como posible candidato a la solución, para esto fue necesario analizar las ventajas que proporciona SQLite sobre el XML en cuanto a capacidad de almacenamiento, seguridad y la facilidad de integración con otra base de datos.

1.10.1. SQLite

En los términos más simples, SQLite es un paquete de software de dominio público que proporciona un sistema de gestión de bases de datos relacionales o RDBMS por sus siglas en inglés (Kreibich, 2010). Creado por D. Richard Hipp, SQLite es una librería de Software programada en lenguaje C, implementa un motor de base de datos SQL transaccional, embebida o autocontenida, sin servidor y sin necesidad de configurar. A diferencia de los motores de base de datos convencionales con la arquitectura cliente-servidor, SQLite es independiente, ya que no se comunica con un motor de base de datos, sino que las librerías de SQLite pasan a integrar la aplicación. La misma utiliza las funcionalidades de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre

procesos (Owens, 2006). En su versión 3, SQLite permite base de datos de hasta dos Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB que almacena gran cantidad de información en un solo campo de la base de datos.

Razones para elegir SQLite como base de datos portable

SQLite tiene una pequeña memoria y una única biblioteca, necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un solo fichero estándar, que puede ser ubicado en cualquier lugar dentro de la jerarquía de directorios. La ventaja de utilizar SQLite consiste en que es mucho más eficiente que XML pues sólo los objetos que se están utilizando se cargarán en la memoria. Además, la búsqueda y filtrado de objetos son más eficientes ya que puede utilizar SQL para encontrar los objetos. La necesidad de transferir los datos hacia otra base de datos hace más sencillo su uso ya que tendrá un solo lenguaje estándar SQL. Además en XML se guarda la información en texto plano, facilitando la lectura de los datos, mientras que en SQLite se necesita de un gestor que permita abrir el fichero, pues su estructura es en formato binario, lo cual impide el entendimiento de los datos. Se escoge SQLite 3.7.6.1, la última versión estable pues la simplicidad de administración, implementación y mantenimiento lo hacen mucho más factible que XML. Se analizará a continuación la API para integrar SQLite con PostgreSQL.

1.11. Java Persistence API

Con esta tecnología se trabajará en la generación de las bases de datos en SQLite y en la integración de esta con PostgreSQL. Para el desarrollo del sistema se utilizará la tecnología de acceso a datos Java Persistence API (JPA), que no es más que una especificación de Sun Microsystems para la persistencia de objetos Java a cualquier base de datos relacional. Esta API fue desarrollada para la plataforma JEE (por sus siglas en inglés, *Java Enterprise Edition*) e incluida en el estándar de EJB 3.0 (por sus siglas en inglés, *Enterprise Java Beans*) (Schnicariol, 2009). La implementación de referencia de JPA es *EclipseLink*. JPA permite al desarrollador trabajar directamente con los objetos en lugar de con sentencias SQL, este se suele llamar proveedor de persistencia. La correlación entre los objetos Java y las tablas de la base de datos se define a través de los metadatos de persistencia de esta forma usará la

información de los metadatos de persistencia, para realizar las operaciones de la base de datos correcta.

La ventaja de esta API es que reduce la cantidad de código necesario para lograr la persistencia permitiendo que la aplicación sea fácil de mantener, así como que favorece la migración de SQLite a PostgreSQL.

A continuación se explicarán los elementos de seguridad a tener en cuenta con el desarrollo del sistema.

1.12. Seguridad en los sistemas informáticos

La seguridad informática es el conjunto de métodos y herramientas destinados a proteger los bienes informáticos de una institución (Garfinkel, y otros, 2003).

La palabra seguridad, en informática, es tan amplia como se quiera pensar, abarca desde temas tan obvios como el resguardar la base de datos de los empleados de una empresa que deben tener accesos a ella, pero muy controlado como el de asegurar que la información mantenga su originalidad.

Un sistema informático es seguro si se puede depender de que se comporte tal y como de él se espera. Garantizar la seguridad de la información es un elemento esencial para poder llegar a obtener un software de calidad. Cuando se habla de seguridad se refiere principalmente a los siguientes términos:

Confidencialidad: Garantiza que la información o los activos informáticos sean accedidos solo por las personas autorizadas para hacerlo.

Autenticación: Garantiza que el acceso a la información sólo puede ser realizado por quienes proporcionan la identidad adecuada.

Integridad: Asegura que el mensaje no ha sido modificado accidental o deliberadamente.

1.12.1. Seguridad en el Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas.

Con el objetivo de garantizar la seguridad del sistema se implementarán algunos mecanismos que avalen la fiabilidad de la información.

A continuación se mencionan los elementos más relevantes en la seguridad del SGI desarrollado:

- Autenticación a la base de datos central.
- Validaciones de la entrada de datos a la base de datos.
- Seguridad en base de datos SQLite.

Uno de los elementos que aportará seguridad al sistema es el uso de SQLite Encrypt el cual se detallará a continuación.

1.12.2. SQLite Encrypt

La extensión de cifrado de SQLite (SEE) es un complemento a la versión de dominio público de SQLite, que permite a una aplicación leer y escribir archivos de base de datos cifrados. El SQLite Encrypt no necesita ninguna instalación ni administración, su encriptación es transparente, fácil de usar y de alta seguridad, ya que utiliza un avanzado algoritmo de encriptación AES *(*por sus siglas en inglés, *Advanced Encryption Standard*) para asegurar que los datos no se puedan descifrar. A partir de 2006, AES es uno de los algoritmos más populares usados en criptografía de clave simétrica (SQLiteEncrypt, 2011).

1.13. Conclusiones del capítulo

Con el desarrollo de este capítulo se determinó que los sistemas de gestión de la información contribuyen de manera positiva a los diagnósticos tecnológicos de entidades legales cubanas. Se definió como metodología de desarrollo de software a Scrum, la cual se modelará con el lenguaje UML, apoyado en la herramienta Visual Paradigm for UML 8.0 Enterprise Edition. Se utilizará el lenguaje de programación Java con el IDE Netbeans 7.1 y el patrón arquitectónico de N- Capas para la implementación del sistema. Para la persistencia de los datos se utilizará JPA, gestionando los datos con el SGBD PostgreSQL 9.1 y SQLite 3.7.6.1 como base de datos portable.

CAPÍTULO 2: CONSTRUCCIÓN DE LA PROPUESTA DE SOLUCIÓN

2.1. Introducción

En el presente capítulo se describe la propuesta de solución de este trabajo y se explican, diseñan e implementan los principales artefactos de la metodología empleada. Igualmente se detalla la arquitectura utilizada, la seguridad y los patrones de diseño empleados.

2.2. Descripción de la propuesta de solución

2.2.1. Funcionamiento del sistema

El sistema fue desarrollado en aras de una mejor reutilización de código y desarrollo en paralelo y con objetivos específicos que tributan a un fin común que es el diagnóstico de la entidad.

El sistema de diagnóstico tecnológico se encarga de la recogida y procesamiento de datos, referentes a la información necesaria para diagnosticar cada entidad. La misma es guardada en una base de datos SQLite, el cual formará parte de la data de todas las entidades diagnosticadas, para luego obtener los diferentes reportes y documentos que facilitarán la toma de decisiones en el futuro proceso de despliegue. También se grafica la infraestructura tecnológica de la entidad y el equipamiento existente con las conexiones entre los equipos, se brindará la opción de exportar lo graficado en formato de imagen PNG (por sus siglas en inglés, *Portable Network Graphics*). Este fichero formará parte del informe de la entidad, para auxiliar la elaboración del informe final y cumplir con los objetivos del diagnóstico.

2.2.2. Despliegue del sistema

El sistema se utilizará primeramente en la entidad por el equipo de diagnóstico para recoger los datos necesarios, se generará un archivo con el gráfico de la infraestructura tecnológica de la entidad y con la información recogida, el cual se trasladará al centro de datos para su posterior procesamiento. Se le dará la opción al cliente de encriptar el archivo para darle seguridad, tema que se abordará en próximos acápites. Luego, a través del mismo sistema,

los archivos generados en la entidad se cargan para almacenar la información en la base de datos central como se muestra en la figura 2, la cual permitirá obtener los reportes que facilitarán la toma de decisiones en el futuro proceso de despliegue.

Figura 2 Despliegue.

2.2.3 Roles involucrados

Para realizar un buen diagnóstico es necesario definir un conjunto de roles especializados que deben estar presentes durante este proceso. Estos roles deben dividirse en dos grupos; roles de la parte cliente, que se refiere a las personas que laboran en las entidades que serán diagnosticadas y roles del equipo de diagnóstico, que se refieren al personal que va a realizar el diagnóstico. En esta sección se identifican cuáles son los roles por cada una de las partes involucradas en el proceso y también se especifican las responsabilidades que deben cumplir los mismos.

Roles de la entidad

Nombre	Descripción	Responsabilidades
Responsable de la entidad	Es la mayor representación en la entidad.	Presentar al equipo de diagnóstico a los demás trabajadores de la entidad.
Administrador	Conoce el funcionamiento y las características de la entidad.	Guiar y ayudar al equipo de diagnóstico en la caracterización de la entidad.

Tabla 1. Roles de la entidad.

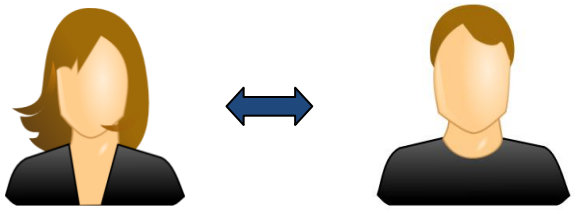

Figura 3 Roles de la entidad.

Roles del equipo de diagnóstico

Nombre	Descripción	Responsabilidades
Jefe del diagnóstico	Es un especialista encargado de dirigir al equipo técnico y tomar las decisiones referentes al diagnóstico.	Realiza los diferentes reportes que permitirán evaluar a la entidad.
Informático	Es un especialista en el diagnóstico que además de ser el jefe del equipo en la entidad visitada, debe recoger la información de la entidad.	Caracterizar de manera general la entidad.
Especialista	Es un especialista en la parte eléctrica y de redes.	Evaluar el estado de las instalaciones eléctricas. Evaluar el estado de la red informática. Evaluar la infraestructura de la entidad. Graficar el equipamiento tecnológico existente.

Tabla 2. Roles del diagnóstico.

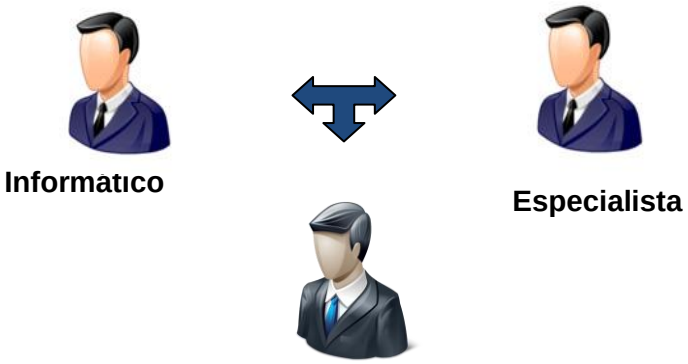

Figura 4 Roles del equipo de diagnóstico.

2.2.4 Definición de la documentación

En cada una de las actividades que se realizan durante el proceso de diagnóstico tecnológico se generan un conjunto de artefactos que son de vital importancia.

En esta sección se describen los documentos a entregarse en el proceso de diagnóstico. De cada uno de los artefactos que se generan durante este proceso se especifica: nombre, descripción y rol responsable de su realización.

Nombre	Descripción	Responsable
Informe de la entidad.	Documento que recoge los reportes impresos con las informaciones generales de la entidad. Recoge además las observaciones obtenidas por el equipo de diagnóstico, y recomendaciones al final del informe.	Jefe del diagnóstico.
Informe final.	Documento que recoge el reporte con la categorización de las entidades, en cuanto a la condición para ser desplegadas.	Jefe del diagnóstico.

Tabla 3. Documentación.

2.2.5 Estrategia de diagnóstico

En esta sección se describen los procesos relacionados con el diagnóstico a un nivel detallado, declarando sus objetivos, responsables, involucrados y documentos de entrada y salida.

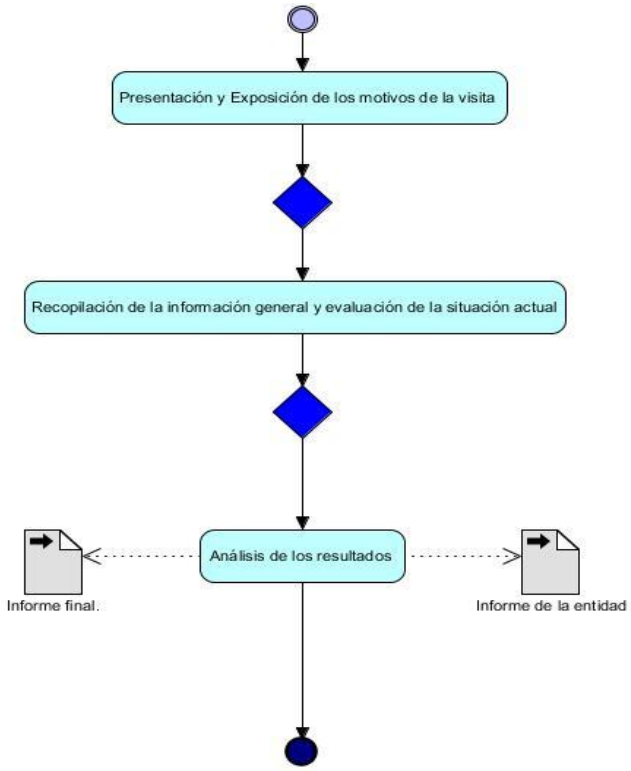

Figura 5. Diagrama de procesos del diagnóstico.

Proceso	**Presentación y exposición de los motivos de la visita.**
Responsable	Equipo de diagnóstico (Jefe de equipo). Responsable en la entidad. Administrador de la entidad.
Objetivos	Realizar la presentación del equipo de diagnóstico y exponer los motivos de la visita.
Involucrados	Equipo de diagnóstico.
Entradas/ Responsable	
Salidas/ Responsable	

Tabla 4. Proceso: Presentación de los motivos de la visita.

Proceso	**Recopilación de la información general y evaluación de la situación actual.**
Responsable	Equipo de diagnóstico (Jefe de equipo y Especialista).

Objetivos	Obtener los datos relacionados con la plantilla del personal y sus características fundamentales como la estructura que presenta la oficina y la cantidad de equipamiento. Además, se determina el estado de las instalaciones eléctricas y de la red local. Se mitigan riesgos y se recomiendan soluciones referentes a los aspectos diagnosticados.
Involucrados	Equipo de diagnóstico.
Entradas/ Responsable	
Salidas/ Responsable	

Tabla 5. Proceso: Recopilación de la información general y evaluación de la situación actual.

Proceso	**Análisis de los resultados.**
Responsable	Equipo de diagnóstico (Jefe de equipo).
Objetivos	Analizar los datos recogidos en el proceso de diagnóstico. Elaborar los informes finales.
Involucrados	Equipo de diagnóstico.
Entradas/ Responsable	
Salidas/ Responsable	Informe final/**Jefe de diagnóstico.** Informe de la entidad/ **Jefe de diagnóstico**
Salidas/ Responsable	

Tabla 6. Proceso: Análisis de los resultados.

Actividades y descripción de los procesos.

A. Presentación y exposición de los motivos de la vista

En esta primera etapa se pretende realizar una presentación adecuada del equipo de trabajo y del personal de la entidad que los atenderá, para posteriormente el jefe del equipo exponer cuáles son los objetivos de la visita, la propuesta de cambio planificada y sus características generales. Este intercambio inicial se debe realizar con la intención de motivar a la dirección de la entidad y a su personal implicado. En esta presentación se deben exponer los aspectos fundamentales a presentar.

Actividades

En esta sección se definen un conjunto de actividades que son importantes para el proceso de presentación y exposición de los motivos de la visita como son:

Exponer los objetivos de la visita: En esta actividad se exponen a los clientes cuáles son los objetivos de la visita.

B. Recopilación de la información general y evaluación de la situación actual.

En este proceso el jefe de equipo debe recopilar toda la información general que pueda caracterizar a la entidad, para esto es necesario realizar un recorrido por la instalación.

En esta etapa el intercambio puede ser de ambas partes y el jefe de equipo puede ayudar y explicar todos los aspectos capturados, de igual modo puede adicionar alguna información que no haya sido analizada y sea de gran importancia para los resultados de la visita.

Actividades

Realizar una caracterización general de la entidad: Se recogen datos referentes a la cantidad de trabajadores y los conocimientos informáticos, áreas de la entidad, así como el equipamiento existente y equipos que se necesitan.

C. Análisis de los resultados.

Se gestionan los datos recogidos durante el proceso de diagnóstico tecnológico. Se clasifica la información recogida en el diagnóstico, realizando un estudio a fondo de cada entidad y evaluando su situación actual. Este es el momento donde se toman las decisiones que favorecerán al despliegue.

Actividades

Estudiar la información recogida: en esta actividad se estudia la información recogida por cada uno de los equipos de diagnóstico en el recorrido por las entidades y se clasifican las mismas.

Generar informes finales: se generan los informes con la información general obtenida en todas las entidades visitadas, así como el informe general de cada una.

Para el desarrollo del sistema es necesario describir y modelar los artefactos que genera la metodología Scrum: el *Producto Backlog* y el *Sprint Backlog*.

2.2.6 Producto Backlog / Pila de tareas del producto

El *Product Backlog* es el inventario de funcionalidades, mejoras, tecnología y corrección de errores que deben incorporarse al producto a través de las sucesivas iteraciones de desarrollo. El *Product Owner* es responsable del contenido, priorización, y disponibilidad de este: nunca se acaba y es usado en la planificación del proyecto, es simplemente una estimación inicial de los requisitos. El *Product Backlog* se desarrolla paralelamente a medida que el producto y el ambiente en el cual se trabaja evolucionan; es dinámico, maneja constantemente los cambios para identificar qué necesita el producto para ser apropiado, competitivo, y útil. Mientras exista un producto, el *Product Backlog* también existe.

A continuación se muestra dicho artefacto, cuenta con el identificador único de la funcionalidad, el nombre, la descripción, el estado y el rol.

ID	Nombre	Descripción	Estado	Rol
Requisitos Funcionales				
	Capturar información general de la entidad	El sistema debe permitir recoger los datos de la identificación de la entidad: ▪ Nombre de la entidad. ▪ Código. ▪ Provincia. ▪ Municipio. ▪ Teléfono fijo de la entidad. ▪ Tipo de oficina. ▪ Observaciones generales.	100%	Informático
2	Capturar datos del departamento de recursos humanos	El sistema debe permitir mostrar los siguientes datos del departamento de recursos humanos: ▪ Cantidad de trabajadores fijos. ▪ Cantidad de trabajadores contratados. ▪ Cantidad de trabajadores que son mujeres. ▪ Cantidad de trabajadores que son hombres. ▪ Cantidad de trabajadores por conocimiento de computación según las categorías: iniciales, ninguno, experto, medio. ▪ Cantidad de trabajadores a capacitar.	100%	Informático
3	Capturar información	El sistema debe permitir recoger los siguientes	100%	Informático

	del funcionamiento de la entidad	datos respecto al funcionamiento de la entidad: ▪ Volumen diario aproximado de trámites. ▪ Volumen mensual aproximado de trámites. ▪ Cantidad de expedientes en el archivo. ▪ Carga de trabajo. ▪ Aplicaciones informáticas existentes. ▪ Servicios informáticos que reciben.		
4	Capturar información del equipamiento informático existente en la entidad	El sistema debe permitir recoger los siguientes datos del equipamiento informático existente en la entidad: ▪ Tipo equipo. ▪ Marca. ▪ Modelo. ▪ Observación. ▪ Cantidad. ▪ Estado. Para los equipos de interconexión: ▪ Tipo equipo. ▪ Marca. ▪ Modelo. ▪ Observación. ▪ Cantidad. ▪ Estado. ▪ Cantidad de puertos Para las computadoras: ▪ Tipo equipo. ▪ Marca. ▪ Modelo. ▪ Observación. ▪ Cantidad. ▪ Estado. ▪ RAM. ▪ Procesador.	100%	Informático
5	Capturar datos generales de la red informática	Si existe red: ▪ Si es cableada o inalámbrica. Si existe Internet: ▪ Proveedor. ▪ Observaciones. Observaciones: ▪ Especialista.	100%	Informático

		▪ Si es desplegable. ▪ Riesgos presentes.		
6	Capturar datos generales del inmueble	El sistema debe permitir recoger los siguientes datos del inmueble: ▪ Tipo de inmueble (propio, alquilado o patrimonio histórico). ▪ Nivel en el que se encuentra ubicado. ▪ Perspectivas de mudanza. ▪ Observaciones.	100%	Informático
7	Capturar características del piso, paredes y techo de la entidad	El sistema debe permitir recoger las características del piso, paredes y techo de la entidad: ▪ Material. ▪ Si tiene filtraciones. ▪ Estado (bueno, malo, regular). ▪ Estado general. ▪ Observaciones.	100%	Informático. Especialista de redes
8	Capturar información para identificar y caracterizar al sistema de seguridad	El sistema debe permitir recoger los siguientes datos del sistema de seguridad: ▪ Si tiene sistema de seguridad. ▪ Tipo de sistema de seguridad: sistema de incendios, dispositivos de vigilancia, sistema de alarma contra intrusos, otros. ▪ Existencia de custodio. ▪ Existencia de rejas. ▪ Si existen salidas de emergencia, cuantas. ▪ Observaciones.	100%	Informático
9	Capturar información para identificar y caracterizar la climatización de la entidad	El sistema debe permitir identificar el estado de la climatización de la entidad: ▪ Cantidad de aires acondicionados y/o ventiladores. ▪ BTU (Cap. refrigeración). ▪ Estado (bueno, regular, malo).	100%	Informático.
10	Capturar información para identificar y caracterizarla acometida telefónica	El sistema debe permitir identificar la acometida telefónica. ▪ Si existe sistema de telefonía: ▪ Ubicación (soterrada, aérea, otras). ▪ Estado (bueno, regular, malo). ▪ Observaciones.	100%	Informático. Especialista de redes.

11	Capturar información para identificar el estacionamiento	El sistema debe permitir identificar del estacionamiento: ▪ Si tiene estacionamiento. ▪ Ubicación. ▪ Observaciones.	100%	Informático.
12	Capturar información para caracterizar las instalaciones eléctricas	El sistema debe permitir capturar los siguientes datos de las instalaciones eléctricas: ▪ Ubicación del transformador. ▪ Capacidad del transformador. ▪ Ubicación del tablero principal. ▪ Ubicación del medidor de corriente. ▪ Capacidad de carga máx. instalada. ▪ Ubicación de los tableros secundarios. ▪ Tipo de corriente: 110v o 220v. ▪ Estado de cableado (bien, regular, mal). ▪ Observaciones. Componentes eléctricos. ▪ Estado de los tableros eléctricos. ▪ Estado del conductor de tierra. ▪ Concentración de carga en los nodos. ▪ Estado técnico de las extensiones eléctricas. ▪ Estado técnico de los empalmes eléctricos. ▪ Estado de los conductores eléctricos (bien, regular, mal). ▪ Estado de los tomacorrientes (bien, regular, mal). Observaciones: ▪ Especialista. ▪ Si es desplegable. ▪ Riesgos presentes.	100%	Informático. Especialista de redes
13	Capturar información del estado de las instalaciones sanitarias	El sistema debe permitir recoger el estado de las instalaciones sanitarias: Estado de las aguas servidas: ▪ Estado (bien, regular, mal). ▪ Observaciones. Estado de las aguas blancas: ▪ Estado (bien, regular, mal). • Observaciones.	100%	Informático
14	Capturar información	El sistema debe permitir identificar la tubería de	100%	Informático

	para identificar y caracterizar tubería de gas	gas: ▪ Si existe tubería de gas: ▪ Ubicación (aérea, soterrada u otros). ▪ Estado (bien, regular, mal). ▪ Observaciones.		Especialista de redes
15	Capturar información de televisión	El sistema debe permitir identificar televisión: • Ubicación. • Ubicación del cableado de la misma: (Aérea, Soterrada u Otros). • Observaciones.	100%	Informático
16	Graficar plano de la entidad	Graficar plano de la entidad con el equipamiento informático.	100%	Informático. Especialista de redes
17	Editar plano de la entidad	Editar plano de la entidad con el equipamiento existente.	100%	Especialista de redes Informático
18	Seguridad al fichero generado	Encriptar y desencriptar fichero generado.	100%	Informático Jefe del diagnóstico.
19	Generar reporte general de la entidad	El sistema debe permitir visualizar reportes en cuanto a la clasificación de las entidades, según la complejidad de las mismas. Clasificadas en cuanto a la cantidad de trabajadores en : ▪ Entidades pequeñas (Hasta 5). ▪ Entidades medianas (Entre 6 y 10). ▪ Entidades grandes (Desde 11 en adelante).	100%	Jefe del diagnóstico
20	Generar reporte con la categorización de la entidad	El sistema debe permitir visualizar reportes en cuanto a la caracterización de las entidades en: entidades desplegables u entidades no desplegables a partir de la situación eléctrica y de redes reportada por especialistas. Las categorías a tener en cuenta son: • Categoría 1: Desplegables sin dificultad. Serán las entidades que están categorizadas por los especialistas como desplegables sin recomendaciones de adecuaciones eléctricas o de red, ni riesgos (sin	100%	Jefe del diagnóstico

		observaciones). • Categoría 2: Desplegables si se hacen adecuaciones. Serán las entidades que están categorizadas como desplegables por los especialistas, pero tienen recomendaciones de adecuaciones eléctrica o de red. • Categoría 3: No desplegables. Entidades categorizadas por especialistas como no desplegables. Este reporte mostrará el tipo de entidad, provincia, municipio y el nombre de los especialistas. El sistema debe permitir una opción para visualizar los detalles de las recomendaciones y riesgos, escritas por los especialistas.		
21	Generar reporte con informaciones generales de la entidad	El sistema debe permitir auxiliar a través de los datos recogidos en cada entidad, la elaboración del informe final. Este reporte mostrará los datos generales de la entidad, la información recogida sobre el equipamiento existente, las instalaciones eléctricas y la red informática existente.	100%	Jefe del diagnóstico

Tabla 7. Product Backlog.

2.2.7 Sprint Backlog / Pila de tareas del Sprint

El *Sprint Backlog* define los trabajos que realizará el equipo durante el sprint para generar el incremento previsto. El equipo crea una lista inicial de estas tareas en la segunda parte del *Sprint Planning Meeting* y asume el compromiso de la ejecución. Solamente el equipo puede cambiar el *Sprint Backlog*. El *Sprint Backlog* es un cuadro altamente visible, es lo que debe conseguir durante una iteración del Sprint. A continuación se tabulan las planificaciones de cada uno de los Sprint creados para implementar el sistema. En él se muestran cuatro aspectos: las tareas que corresponde al Sprint en cuestión; el sistema a la que responde; el estado, en este se indica el porciento de cumplimiento de cada tarea; el tiempo estimado y el real para realizar cada una de las tareas (en días).

Pila de Sprint I (Bases de datos)				
Backlog Tarea	**Sistema de diagnóstico tecnológico**	**Estado**	**Estimado**	**Real**
Diseño y Generación de la bases de datos				
Diseñar el diagrama Entidad-Relación	Sistema	100%	5	4
Diseñar el diagrama de clases persistentes	Sistema	100%	5	2
Generar la base de datos en PostgreSQL	Sistema	100%	20	15
Generar la base de datos en SQLite	Sistema	100%	15	15

Tabla 8. Pila de Sprint I (Bases de datos).

Sprint Goal

El objetivo de este Sprint es diseñar y construir las bases de datos.

Con el término del Sprint I se tiene logrado el 20% de la solución total puesto que sus funciones son vitales para el funcionamiento del sistema.

Pila del Sprint II (Construcción del Sistema de diagnóstico tecnológico)			
Backlog Tarea	**Estado**	**Estimado**	**Real**
Diseño e Implementación de formularios y diagramas			
Formulario Principal (Sistema de diagnóstico tecnológico)	100%	7	6
Formulario Gráfico	100%	30	30
Formulario Información general de la entidad	100%	4	4
Formulario Información de recursos humanos	100%	4	4
Formulario Información sistemas y/o servicios informáticos existentes	100%	4	4
Formulario Información del funcionamiento informático	100%	4	4
Formulario Información del equipamiento informático existente	100%	4	4
Formulario Información del inmueble	100%	4	4

Formulario Información de los pisos, paredes y techos de la entidad	100%	4	4
Formulario Información de la seguridad de la entidad	100%	4	4
Formulario Información de la climatización de la entidad	100%	4	4
Formulario Información de la acometida telefónica	100%	4	4
Formulario Información estacionamiento	100%	4	4
Formulario Información de instalaciones eléctricas de la entidad	100%	4	4
Formulario Información de instalaciones sanitarias de la entidad	100%	4	4
Formulario Información de la tubería de gas	100%	4	4
Formulario Información de la televisión	100%	4	4
Validar de la entrada de datos por el usuario	100%	15	15
Validar de la entrada de datos a la base de datos	100%	25	25
Componente para encriptar y desencriptar fichero de base de datos portable	100%	30	30
Diseñar diagramas de clases del diseño de la capa Negocio. Paquete Clases	100%	2	2
Diseñar diagramas de clases del diseño de la capa Negocio Paquete Nomencladores	100%	2	2
Diseñar diagramas de clases del diseño de la capa Presentación. Paquete Visual	100%	2	2
Diseñar diagramas de clases del diseño de la capa Negocio Paquete Clases-Grafico			
Diseñar diagramas de clases del diseño de la capa. Acceso a Datos. Paquete DAO	100%	2	2

Tabla 9. Pila del Sprint II (Construcción del Sistema de diagnóstico tecnológico).

Sprint Goal

El objetivo de este Sprint es realizar todas las actividades que del *Product Backlog* tributan a la obtención y gestión de la información.

El Sprint II deja completamente funcional el Sistema de diagnóstico tecnológico. Con este se puede lograr procesar toda la información necesaria para diagnosticar la entidad. Se garantiza el cumplimiento del 100 % de las funcionalidades críticas del sistema propuesto.

Pila del Sprint III (Validación)			
Backlog Tarea.	**Estado**	**Estimado**	**Real**
Validación del diseño a través de métricas de diseño	100%	10	10
Pruebas de caja negra	100%	15	15
Pruebas de caja blanca	100%	15	15

Tabla 10. Pila del Sprint III (Validación)

Sprint Goal

El objetivo de este Sprint es realizar las pruebas que validan el sistema demostrando la fiabilidad del mismo.

En este Sprint es donde se documenta y prueba el correcto cumplimiento de las funcionalidades planteadas en el *Product Backlog* y se verifica el diseño a través de métricas validando así el diseño del sistema.

Existen también requisitos adicionales que la aplicación debe tener asociados al cumplimiento de las funcionalidades divididas en Sprint. Los requisitos no funcionales no se refieren a funciones específicas que proporciona el sistema. Son restricciones de los servicios o funciones ofrecidas por el sistema (fiabilidad, tiempo de respuestas, capacidad de almacenamiento, etc.). Generalmente se aplican al sistema en su totalidad (Málaga, 2012).

2.2.8 Requisitos adicionales.

Apariencia o interfaz externa:

Diseño sencillo, una interfaz simple de usar e interactiva para que al usuario le sea fácil el trabajo con el sistema.

Usabilidad:

La baja complejidad del sistema hace que pueda ser utilizado por personas con poca o experiencia, pues sus interfaces resultan agradables a la vista y de fácil entendimiento.

Software:

- El servidor de base de datos debe ser PostgreSQL (9.1).
- En la máquina donde se ejecutará el sistema, se deberá encontrar instalada la máquina virtual de java en versión 1.6 o superior a esta.

Hardware:

- Es necesario tener un mínimo de 128 MB de RAM, recomendándose que se tengan 256 MB o más.
- El procesador en principio no es tan crítico como la memoria RAM, pero se recomienda utilizar, en cuanto a la velocidad de procesamiento, 1.33 GHz en adelante.

2.3 Construcción del sistema.

2.3.1 Patrón arquitectónico N- Capas

Una arquitectura de software diseñada en capas consiste en la definición de niveles de abstracción, los cuales tienen una función específica permitiendo un diseño modular. Ello permite que se creen sistemas con un bajo acoplamiento entre sus módulos o componentes. Una variante de este patrón muy utilizada es la de tres capas. El cual se fracciona al sistema informático en la capa de Presentación, la capa de lógica del Negocio y la capa de Acceso a Datos.

Una restricción de este patrón consiste en que las capas inferiores no deben de conocer ni hacer llamadas a procedimientos implementados en capas superiores, sino que las funcionalidades que ellas ofrecen son accedidas desde niveles mayores; pero esto también puede ser una gran ventaja (Reynoso, 2004). Para el desarrollo de la solución se adoptará una arquitectura de tres capas debido sus potencialidades para la reutilización y el aprovechamiento de los beneficios de una clara separación entre las funciones desplegadas en capas. Además, permite la estandarización y proporciona una distribución clara del trabajo entre los miembros de un equipo de desarrollo.

El sistema estará divido en tres capas fundamentales:

- Presentación.
- Negocio.
- Acceso a Datos.

Figura 6. Patrón arquitectónico en tres capas.

Se decidió utilizar este estilo debido a que permite organizar la estructura lógica de un sistema en capas separadas, con responsabilidades distintas y relacionadas, con una separación clara y cohesiva de intereses.

La **capa de Presentación** posee interfaces que interactúan con el usuario, las cuales permiten la recogida de información referente a las entidades, así como la interfaz que permite graficar la infraestructura tecnológica. Además, contiene formularios para mostrar los distintos reportes que apoyen a la toma de decisiones.

La **capa de Negocio** es la encargada de recibir las peticiones generadas en la capa de presentación y transmitirlas a la capa de gestión de datos en caso necesario, es donde reside la lógica de negocio. Contiene las clases a persistir, y todas las clases que implementan las funcionalidades para graficar, utilizando para su desarrollo la biblioteca *Visual Library API 2.0* y

el *Framework Swing*, que posibilita una interfaz más amigable para la interacción con el usuario.

En la **capa de Acceso a Datos** residen los datos de forma permanente en el sistema y es la encargada de nutrir a la capa del negocio con la información solicitada. Para lograr un mayor nivel de abstracción en la persistencia de los datos, se utilizó JPA. Esta capa contiene las clases que interactúan con las bases de datos, las cuales permiten realizar operaciones sobre las mismas de forma transparente para la capa de negocio.

2.3.2 Visual Library API 2.0

El *Visual Library API* de Netbeans es una biblioteca gráfica de alto nivel. Está diseñada para apoyar aplicaciones que necesitan mostrar diagramas o gráficos editables.
La API proporciona un conjunto de paquetes reutilizables. Cada paquete contiene un conjunto predefinido de *widgets*, modelos y acciones. Un *widget* es muy similar al *JComponent* de *Swing*, se puede crear una visualización de una manera sencilla y flexible. Cada control tiene, construido en él, diversas propiedades. (Böck, 2011)

2.3.3 Patrones de diseño utilizados en el desarrollo del sistema.

Visto desde el desarrollo del marco de trabajo se han utilizado algunos de estos patrones de diseño que resultarán de gran importancia para el mismo ya que permiten la reutilización del diseño que no es despreciable, lo cual provee de numerosas ventajas en cuanto a la reducción de esfuerzo de desarrollo y mantenimiento, mejora en la seguridad informática, eficiencia y consistencia de los diseños. Estos, elevan además la flexibilidad, modularidad y extensibilidad: factores internos e íntimamente relacionados con la calidad percibida por el usuario.

De los patrones GOF se hizo uso:

Método Plantilla (Template Method): Este patrón se evidencia en la clase *Reporte* en el método *crearDocumento*, permitiendo a las subclases (*ReporteOficina* y *ReporteFinal*) redefinir el funcionamiento de los pasos de este método, sin cambiar la estructura del mismo.

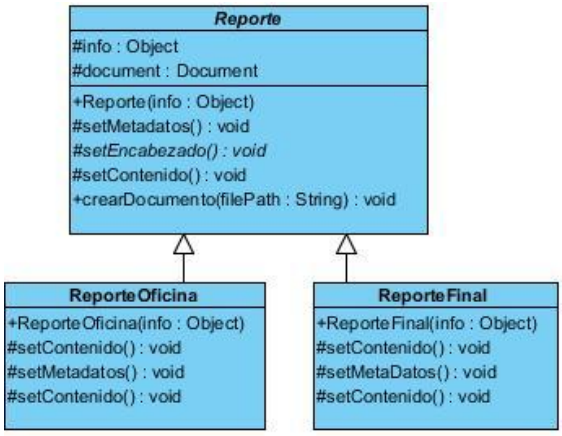

Figura 7. Patrón Método Plantilla.

Singleton: Se utiliza el patrón implementado en JPA del *EntityManager*, permite asignarle un *EntityManagerFactory* el cual representa una unidad de persistencia particular.

También se hizo uso de los patrones GRASP:

Experto: Asigna una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad. Se utiliza por ejemplo, en la clase *Oficina* de la capa del Negocio al implementar los métodos de acceso de la información que ella conoce.

.

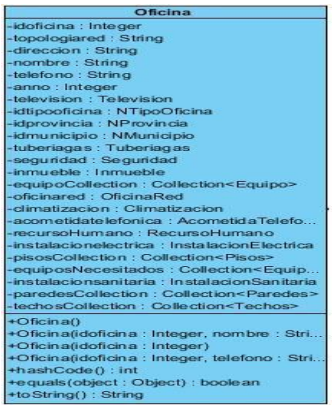

Figura 8. Patrón Experto.

Creador: En el patrón creador se asigna la responsabilidad de que una clase B cree un objeto de la clase A si se cumple uno o más de los casos siguientes (respondiendo al problema de quién debería ser el responsable de una nueva instancia de una clase):

- B agrega objetos de A.
- B contiene objetos de A.
- B registra instancias de objetos de A.
- B utiliza más estrechamente objetos de A.
- B tiene los datos de inicialización que se pasarán a un objeto A cuando sea creado (por tanto B es un Experto con respecto a la creación de A).

Se utiliza en las clases de la capa de Negocio al agregar en la clase *Oficina* objetos de las demás clases presentes en esta capa.

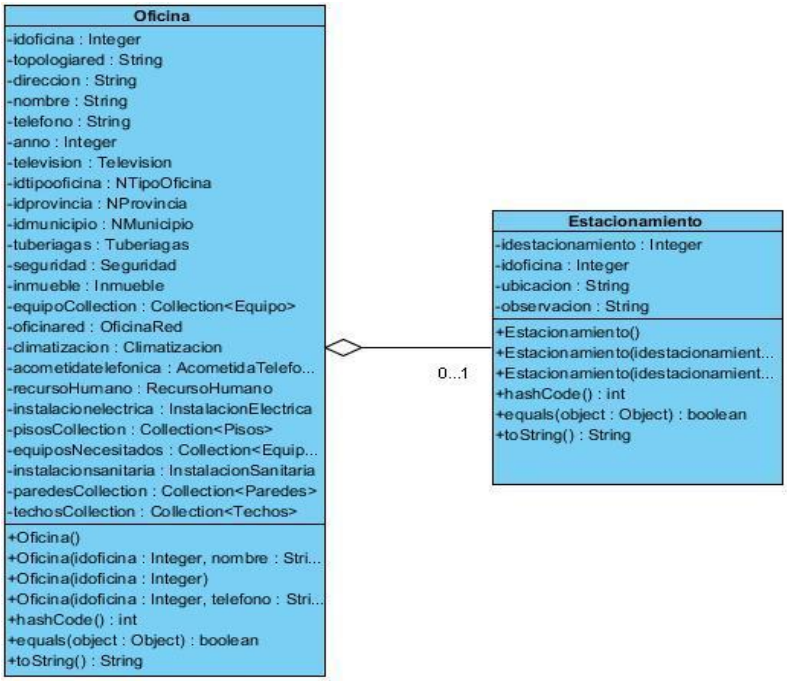

Figura 9 Patrón Creador.

Controlador: Se evidencia en la clase *OficinaController*. Esta clase sirve como intermediaria entre la interfaz del formulario principal y las clases subyacentes, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado de la capa de Negocio.

Figura 10 Patrón Controlador.

Alta Cohesión: Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas para que no realicen un trabajo enorme. En todas las clases se implementa la funcionalidad de este patrón, principalmente en la clase *OficinaDAO* donde se mantiene la complejidad dentro de límites manejables.

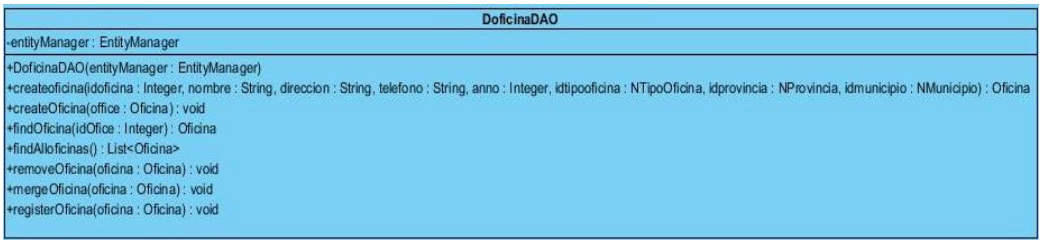

Figura 11 Alta Cohesión.

2.3.4 Seguridad.

Una vez recopilada la información del diagnóstico tecnológico en la entidad legal, esta es guardada en una base de datos portable, la cual puede ser enviada por vía email o en los mejores casos entregada personalmente. A continuación se explican los elementos más relevantes en la seguridad del SGI desarrollado.

Seguridad en la base de datos SQLite.

Se le dará la opción al cliente de encriptar el archivo, recomendando siempre que tenga en cuenta los posibles ataques que pueden ocurrir como: la apropiación del archivo por personal ajeno, atentando contra la confidencialidad de la información, la modificación de los datos,

atentando contra la integridad como por ejemplo: la inclusión de campos o tablas en la base de datos, inclusión de virus, entre otros. Todos estos posibles ataques afectan la fiabilidad de los datos a la hora de generar los reportes que faciliten la toma de decisiones.

En el caso de que el archivo sea encriptado se utilizará la extensión de cifrado SQLite Encrypt. Luego de haber finalizado la recogida de la información se procederá a encriptar el fichero.

Funcionamiento del componente para encriptar:

Una vez seleccionado el fichero a encriptar se procederá a insertar la contraseña para encriptarlo; la misma es encriptada mediante el algoritmo MD5, el cual no permite desencriptarla. Esta contraseña encriptada es guardada en un fichero aparte, que tiene como composición el nombre del fichero entrado por el usuario más la terminación “passEnc”, ej. “InformaciónpassEnc”. De esta forma el fichero quedará encriptado añadiendo al nombre la terminación “Enc”, ej “InformaciónEnc”.

Para desencriptar el fichero se insertará nuevamente la contraseña, la cual es encriptada y comparada con la que está guardada, permitiendo (en caso de coincidir) desencriptar el fichero.

Validaciones de la entrada de datos de la entidad a la base de datos portable.

Se valida la recogida de la información de la entidad comprobando que no se introduzcan datos incorrectos en el sistema por ejemplo en los campos donde solo se admitan letras se valida que no entren números o caracteres extraños. Para el caso de la plantilla de recursos humanos se verifica que el total de trabajadores fijos y contratados coincida con el total de hombres y mujeres, para que los reportes tengan mayor calidad.

Autenticación de la entrada de datos por el usuario en el centro de datos.

Se valida la entrada de datos a la base de datos central desde el sistema con la autenticación de un único usuario, el jefe de diagnóstico, el cual puede acceder desde cualquier máquina con conexión hacia la base de datos central, una vez autenticado podrá cargar en el sistema las entidades a las que se les hizo el diagnóstico y guardarlas para obtener los reportes que

faciliten a la toma de decisiones.

2.3.5 Diagramas de clases de diseño por Sprint Backlog.

Sprint I

Diagrama de clases persistentes (ver anexo, figura 1).

Diagrama Entidad - Relación

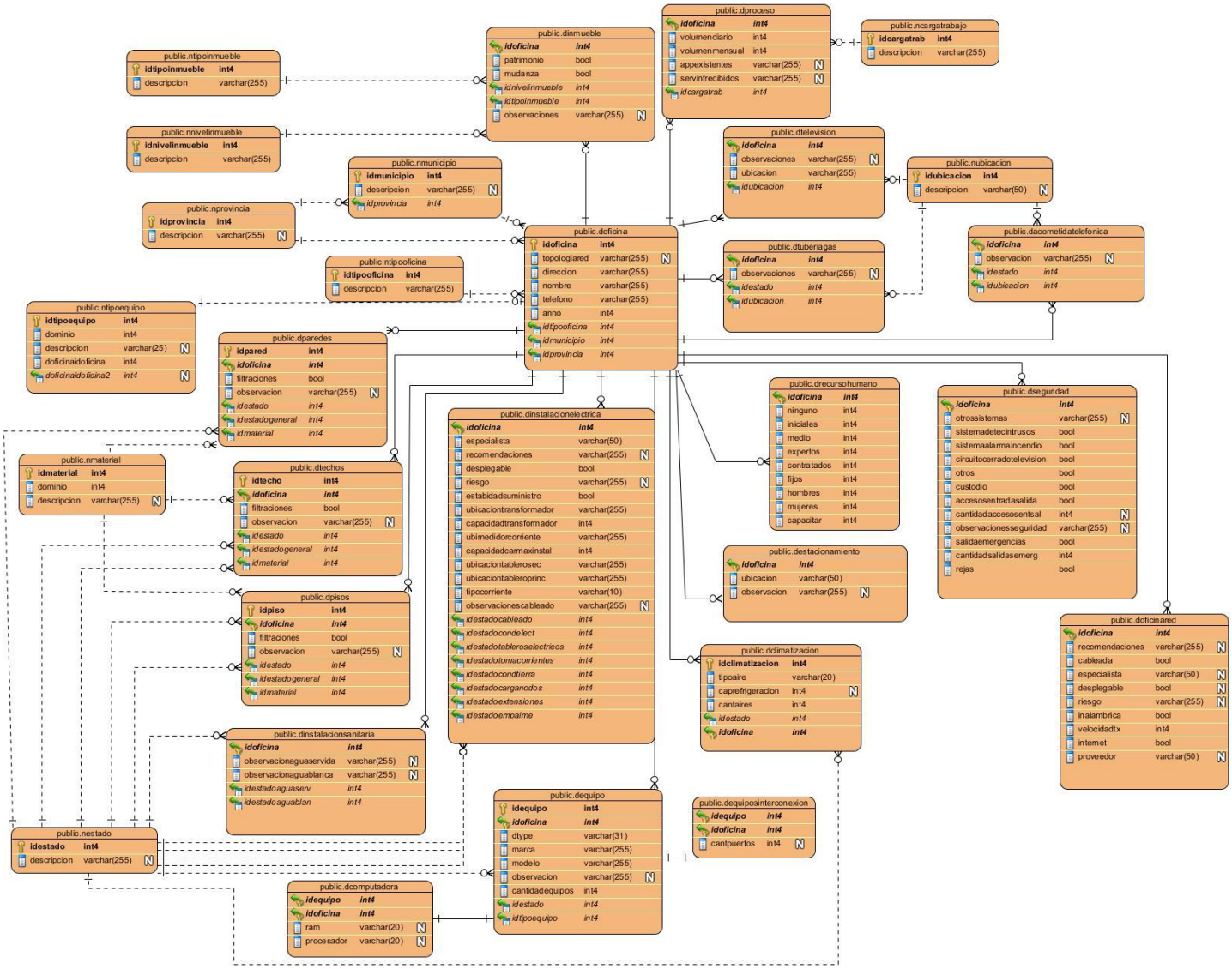

Figura 12 Diagrama Entidad-Relación - Sprint I

Sprint II

El diagrama de clases de diseño muestra el patrón arquitectónico utilizado en la aplicación, está fraccionado en paquetes que encapsulan el diagrama de clases del diseño de la capa de Presentación Sprint II, el diagrama de clases del diseño de la capa de Negocio Sprint II y el diagrama de clases del diseño de la capa de Acceso a Datos Sprint II.

Diagrama para la capa de Presentación. Muestra todos los formularios que se utilizan en la captura de información. De estos se describen dos diagramas separados por paquetes:
Diagrama de clases del diseño del paquete Visual (ver anexo, figura 2).
Diagrama de clases del diseño del paquete Visual – Gráfico (ver anexo, figura 3).

Diagrama para la capa de Negocio. Muestra toda la lógica del negocio empleada para la captura de la información. De estos se describen cuatro diagramas separados por paquetes:
Diagrama de clases del diseño del paquete Clases.
Diagrama de clases del diseño paquete Nomencladores (ver anexo, figura 4).
Diagrama de clases del diseño del paquete Reportes (ver anexo, figura 5).
Diagrama de clases del diseño del paquete Clases-Gráfico (ver anexo, figura 6).

Diagrama para la capa de Acceso a datos. Muestra las clases de acceso a datos utilizadas en este Sprint (ver anexo, figura 7).

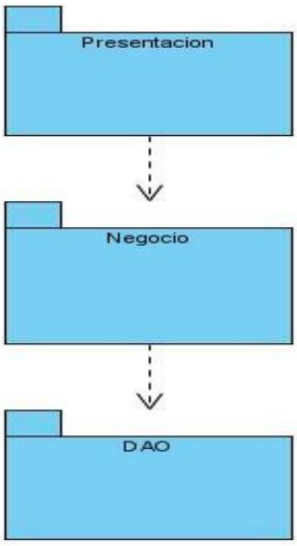

Figura 13 Capas del sistema.

Diagrama de clases del diseño de la capa del Negocio. Paquete Clases.

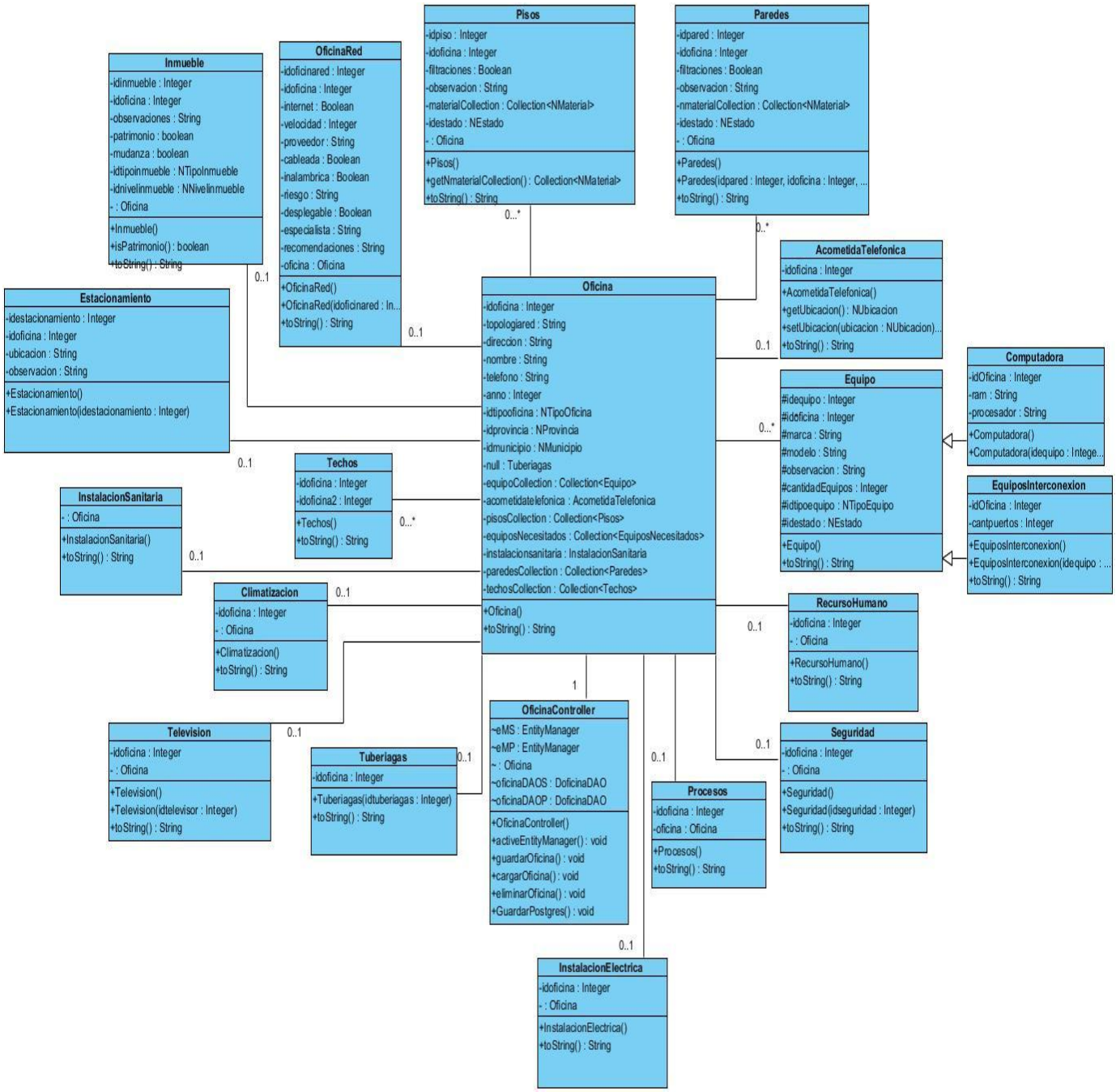

Figura 14 Diagrama de clases de diseño de la capa del Negocio. Paquete Clases-Sprint II.

2.3.6 Tratamiento de excepciones.

Es de vital importancia para lograr el correcto funcionamiento de cualquier sistema, identificar y controlar los posibles errores que se pueden presentar a la hora de interactuar con el software. Para el correcto funcionamiento de la aplicación se tratarán estos errores de forma tal que las interacciones con la base de datos se realicen de forma correcta, para lograr esto se validaron los datos a tratar; además en los formularios se insiste en que el usuario introduzca la menor cantidad posible de datos, aprovechando al máximo los campos calculables dentro del formulario, controles de selección como: botones de opción (*JRadioButton*), casillas de verificación (*JCheckBox*), y listas de selección (*JComboBox*), entre otros. De esta forma el usuario selecciona entre opciones predefinidas lo que no da margen al error. En el caso de la entrada de datos por parte del usuario se muestran mensajes que ilustran la incorrecta inserción en caso de existir errores, modificación o mala manipulación de datos en general con las posibilidades que brinda el lenguaje Java mostrando al usuario una explicativa sobre el posible error cometido. Se implementa una segunda validación de los datos enviados por el usuario y recibidos por la base de datos para lograr minimizar a cero los posibles errores y lograr que el usuario interactúe con un sistema de calidad.

2.3.7 Estándares de codificación.

A continuación se analizarán los estándares o convenciones de programación empleados en el desarrollo del software. Este sistema está basado en los estándares recomendados por Sun Microsystems, que han sido difundidos y aceptados ampliamente por toda la comunidad Java, y que han terminado por consolidarse como un modelo estándar de programación de facto. Estas normas son muy útiles por muchas razones, entre las que destacan:

- Facilitan el mantenimiento de una aplicación. Dicho mantenimiento constituye el 80% del coste del ciclo de vida de la aplicación.
- Permiten que cualquier programador entienda y pueda mantener la aplicación ya que puede que no sea por sus autores.
- Los estándares de programación mejoran la legibilidad del código, al mismo tiempo que permiten su compresión rápida.

Los convenios de código para el lenguaje de programación Java fueron revisados y actualizados el 20 de abril de 1999.

Organización de ficheros

- Fichero fuente Java (.java): Cada fichero fuente Java debe contener una única clase o interfaz pública.

Sangría

Como norma general se establecen 4 caracteres como unidad de sangría, en el caso de la aplicación en cuestión se hizo uso de facilidades para formatear código Java del IDE NetBeans.

- Longitud de línea: La longitud de línea no supera los 80 caracteres por motivos de visualización e impresión.

Declaraciones

Declaración de clases / interfaces: Durante el desarrollo de clases / interfaces se siguieron las siguientes reglas de formateo:

- No incluir ningún espacio entre el nombre del método y el paréntesis inicial del listado de parámetros.
- El carácter inicio de bloque ("{") debe aparecer al final de la línea que contiene la sentencia de declaración.
- El carácter fin de bloque ("}") se sitúa en una nueva línea tabulada al mismo nivel que su correspondiente sentencia de inicio de bloque, excepto cuando la sentencia sea nula, en tal caso se situará detrás de "{".

Nomenclatura de identificadores

- Se utilizarán nombres descriptivos, destacando el significado antes que la tipología del parámetro. En este caso se utiliza la capitalización Camel para el nombre de los atributos, variables y métodos.
- La capitalización Pascal se utilizará para los nombres de clases y las interfaces. Si el nombre es compuesto, cada palabra componente comienza con mayúsculas. Los nombres son simples y descriptivos.

Prácticas de programación

Los atributos serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas, en tales casos serán declarados como protegidos. El acceso a los atributos de una clase se realizará por medio de los métodos "get" y "set" correspondientes.

2.3.8 Conclusiones del capítulo

En el presente capítulo se describió el funcionamiento de la solución propuesta, lográndose completar al 100% las funcionalidades del sistema previstas en el Product Backlog, cumpliendo de esta forma con las necesidades del cliente. Se definió e implementó la arquitectura de N - Capas con las capas de Presentación, Negocio y Acceso a Datos, así como los patrones de diseño GOF, de estos el Método Plantilla y el Singleton y de los GRASP se utilizaron el Experto, el Creador, el Controlador y el de Alta Cohesión. Se describió y modeló cada Sprint desarrollado, tomando para esto los artefactos generados por cada ciclo y de acuerdo a las características de cada uno, entre ellos se encuentran los Diagramas de las Clases de Diseño, Clases Persistentes y Modelo Entidad - Relación. Además se realizó una valoración de los elementos que darán seguridad al sistema: validaciones de entrada de datos a la base de datos portable, autenticación de un único usuario a la base de datos central, seguridad en la base de datos portable SQLite, así como el uso de SQLite Encrypt, que permitirá cifrar y descifrar el fichero portable.

CAPÍTULO 3: VALIDACIÓN DEL SISTEMA

3.1. Introducción

La definición estándar de calidad en ISO-8402 es: “La totalidad de rasgos y características de un producto, proceso o servicio que sostiene la habilidad de satisfacer estados o necesidades implícitas”. Es por eso que en el presente capítulo se valida la calidad del sistema. Para esto se ejecutan pruebas dirigidas al software, con el objetivo de medir el grado en que este cumple con los requerimientos. En las pruebas se usan casos de prueba, especificados de forma estructurada mediante técnicas de prueba. Además se realizaron métricas para validar el diseño del sistema en cuestión.

3.2. Validación.

3.2.1. Métricas de validación del diseño.

El IEEE “*Standard Glosary of Software Engering Terms*” define como métrica: “Una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado*”.*

Alguno de los elementos que permiten evaluar la calidad del diseño son las métricas de validación del diseño.
Existen varios tipos de métricas orientadas a objetos como son:

- Métricas orientadas a las operaciones.
- Métricas para pruebas orientadas a objeto.
- Métricas sobre el tamaño del software.
- Métricas orientadas a clases.

En este último caso existen varios tipos de series aplicadas, se pueden citar algunas como las de Chidamber y Kemerer (CK) y Lorentz y Kidd (LK). En el diseño desarrollado se aplicaron algunas de las métricas propuestas por las series CK y LK, por ser las más reconocidas y las que más se ajustan al diseño desarrollado. Los objetivos principales de las métricas orientadas a objetos son los mismos que los existentes para las métricas surgidas para el software estructurado:

- Comprender mejor la calidad del producto.
- Estimar la efectividad del proceso.

- Mejorar la calidad del trabajo realizado en el nivel del proyecto.

En el libro de métricas realizado por Lorenz y Kidd, se dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase X se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema X en su totalidad (Laranjeira A. , 1990).

Para la validación del diseño del software se escogió la métrica de la primera serie (LK):

- **Tamaño de Clases (TC):** Las clases pueden medirse determinando el total de operaciones, tanto heredadas como privadas de la instancia que se encapsulan dentro de una clase, más el total de atributos, atributos tanto heredados como privados de la instancia encapsulados por la clases.

  Los valores grandes para esta métrica, indican que la clase posee bastante responsabilidad y complejidad, esto reduce la reutilización de esta clase y dificulta las pruebas sobre la misma.

  Para el funcionamiento de esta métrica se pueden calcular los promedios para el número de atributos y operaciones de clase.

  El tamaño general de una clase se puede determinar empleando las medidas siguientes:

  - El número total de operaciones.
  - El número de atributos.

  Teniendo en cuenta las medidas de referencia en lo que respecta al número de operaciones y/o atributos de las clases se establece que: un tamaño de clase pequeño es aquel que tiene un valor menor o igual que 20, un tamaño de clase medio es aquel cuyos valores exceden a 20 y son menores o iguales que 30 y un tamaño de clase grande es aquel que es mayor que 30.

  El Tamaño Operacional de Clase (TOC) está dado por el número de métodos asignados a una clase. La cual afecta un grupo de atributos como se muestra a continuación:

Atributo que afecta	**Modo en que lo afecta**
Responsabilidad	Un aumento del TOC implica un aumento este atributo.

Complejidad	Un aumento del TOC implica un aumento este atributo.
Reutilización	Un aumento del TOC implica una disminución de este atributo.

Tabla 11 Atributos de Tamaño Operacional de Clases.

Atributo	Categoría	Criterio
Responsabilidad	Pequeña	<= Promedio de Procedimientos
	Media	>Procedimiento <= 2 * Procedimiento
	Grande	> 2 * Procedimiento

Tabla 12 Umbral de Responsabilidad.

Atributo	Categoría	Criterio
Complejidad	Pequeña	<= Promedio de Procedimientos
	Media	>Procedimiento <=2 * Procedimiento
	Grande	>2 * Procedimiento

Tabla 13 Umbral de Complejidad.

Atributo	Categoría	Criterio
Reutilización	Pequeña	>2* Procedimiento
	Media	>Procedimiento <=2 * Procedimiento
	Grande	<= Promedio de procedimientos

Tabla 14 Umbral de Reutilización.

Tomando como referencia las clases de la lógica de negocio del sistema, se cuenta con un total de 40 clases (ver anexos, tabla 1), con un promedio de procedimientos de aproximadamente 7.

Después de aplicar esta métrica al diseño de la solución propuesta se llegó a las siguientes conclusiones. Los valores de tamaño de clase quedan distribuidos de la siguiente manera teniendo en cuenta tres criterios o atributos de calidad:

- Responsabilidad (ver anexo, figura 8).
- Complejidad (ver anexo, figura 9).
- Reutilización (ver anexo, figura 10).

Con como resultado de esta métrica se obtuvo que el 50% de las clases tienen responsabilidad y complejidad pequeña indicando con esto que tienen un pequeño tamaño de clase. Además el 50% de las clases tienen una alta reutilización indicando con esto que tienen un pequeño tamaño de clase.

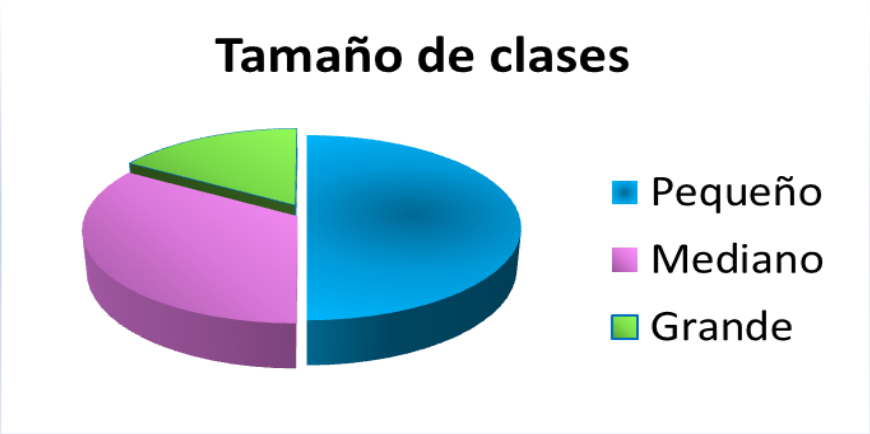

Figura 15 Tamaño de las Clases

Resultados: Con la figura 15 se concluye la realización de esta métrica interpretando los valores asociados a esta, lo cual deja evidenciado que el TC del sistema desarrollado es relativamente pequeño cumpliendo con cado uno de los elementos a tener en cuenta para desarrollarla, proporcionando esto, un diseño de baja responsabilidad y complejidad, alta reusabilidad y fácil de probar.

De la serie CK se emplearon las siguientes métricas:

- **Número de Descendientes (NDD):** Un mayor número de hijos representa una mayor reutilización de código, pero presenta inconvenientes como una mayor probabilidad de usar incorrectamente la herencia, creando abstracciones erróneas; es decir, existe una posibilidad de que algunos descendientes no sean miembros, realmente apropiados, de la clase predecesora. Además, dificulta la modificación de esta clase, ya que afecta a

todos sus hijos. Todo esto trae consigo que se requiera mayor número de recursos para probar.

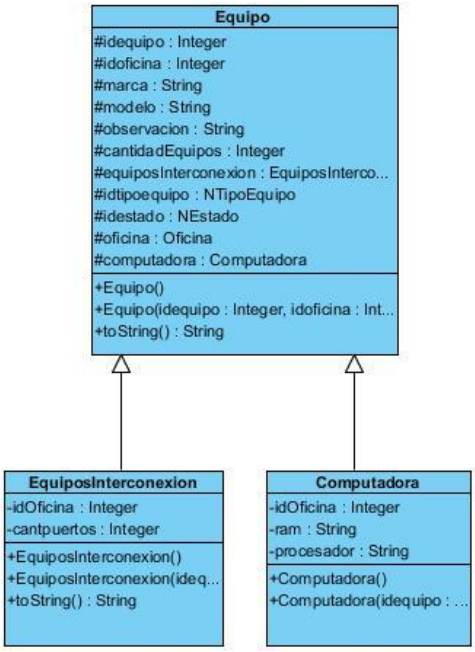

Figura 16. Descendencia mayor existente.

Resultado: Esta métrica aportó como resultado que el número máximo de descendientes es dos (ver anexo, figura 11). Siendo este positivo, pues permite la reutilización, favorece la abstracción, permite la modificación moderada y facilita las pruebas en cuestión.

- **Árbol de Profundidad de Herencia (APH)**: Altos niveles de herencia indican objetos complejos, los cuales pueden ser difíciles de probar y mantener, esto se debe a que cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos. Por el lado positivo, los valores de APH grandes implican un gran número de métodos que se reutilizarán. (Pressman, 2001)
  Según la bibliografía consultada se define como una profundidad negativa, el nivel 6.
  Resultado: Esta métrica aportó como resultado que el nivel más alto de herencia entre las clases del diseño es dos, esto determina que el diseño no es complejo, es fácil de probar, adaptar y modificar. La figura 17 muestra el nivel de herencia.

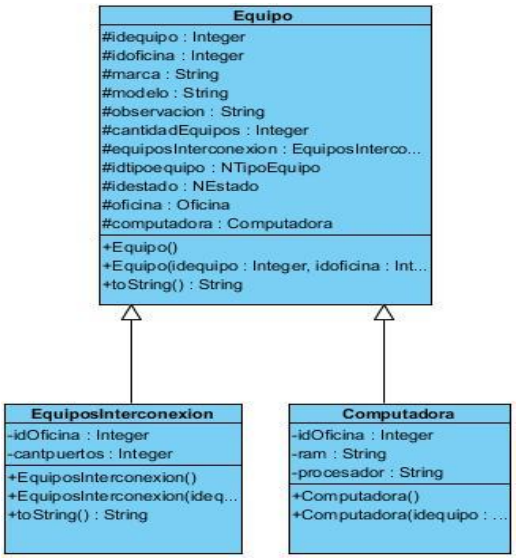

Figura 17 Nivel de herencia.

Para un nivel de herencia igual a dos se evidencia poca reutilización lo que facilita el diseño y el mantenimiento del sistema (ver anexo, figura 12).

3.2.2. Pruebas de Caja Negra.

Estas pruebas se enfocan en los requerimientos establecidos y las funcionalidades del sistema, se realizan sobre las interfaces del sistema para demostrar que cada funcionalidad es correcta. Las pruebas de caja negra se centran en los requisitos funcionales del software.

Para realizar estas pruebas fueron diseñados 25 casos de prueba, el equipo de calidad de la facultad 3 fue el encargado de realizar las pruebas de caja negra al presente sistema, la técnica empleada por el equipo fue:

Partición equivalente: Divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. (Pressman, 2001)

En el diseño de casos de prueba para partición equivalente se procede en dos pasos:

Se identifican las clases de equivalencia. Estas son identificadas tomando cada condición de entrada (generalmente una oración o una frase en la especificación) y repartiéndola en dos o más grupos.

- Es de notar que dos tipos de clases de equivalencia están identificados: las clases de equivalencia válidas representan entradas válidas al programa, y las clases de equivalencia inválidas que representan el resto de los estados posibles de la condición (es decir, valores erróneos de la entrada).
- Se define los casos de prueba. El segundo paso es el uso de las clases de equivalencia para identificar los casos de prueba. En el proceso se asigna un número único a cada clase de equivalencia. Hasta que todas las clases de equivalencia válidas han sido cubiertas por los casos de prueba, se escribe un nuevo caso de prueba que cubra la clase de equivalencia válida. Y por último se escribe un caso de la prueba cada una de las clases de equivalencia inválidas descubiertas (Glenford , 2004).

A continuación se muestra la descripción de uno de los casos de prueba diseñados por el equipo de desarrollo.

Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**
SC 1: Identificar la entidad.	EC 1.1 Seleccionar la opción Guardar.	Al insertar los datos requeridos se selecciona la opción guardar, para almacenar la información.
	EC 1.2: Seleccionar la opción Cancelar.	Mediante este escenario se cancela la interfaz.
	EC 1.3: Existen datos incorrectos.	Al insertar datos, si existe algún campo que no tiene el tipo de dato requerido se muestra un ventana de error.

Tabla 15 Sección Identificar la entidad.

Escenario	**Nombre**	**Cód.**	**Prov.**	**Mun.**	**Teléf.**	**Tipo de entidad**	**Observ.**	**Resp.**
EC 1.3: Existen datos incorrectos.	Tribunal Supremo V	C48 V	La Hab. V	C. de La Hab. V	286504 V	1 F	-	Muestra un mensaje de error.
	Tribunal Supremo V	C48 V	La Hab. V	C. de La Hab. V	28625o 4 *F*	Tribunal Supremo V	-	Muestra un mensaje de error.

EC 1.2: Seleccionar la opción Cancelar.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Regresa a la interfaz principal.
EC 1.3: Seleccionar la opción Guardar.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Muestra una ventana informando que los datos han sido guardados correctamente.

Tabla 16 Sección Identificar entidad.

Resultados: Se diseñaron casos de prueba, obteniendo resultados satisfactorios en cada uno, razón que avala la aceptación funcional del sistema por parte del equipo de calidad de la facultad 3. Este proceso se realizó cubriendo tres iteraciones, logrando en cada caso disminuir las No Conformidades (NC). Para la primera iteración se determinaron 12 NC, en la segunda 3 NC y en la tercera se liberó el sistema, ya que no fueron detectadas NC como se muestra en la figura 18.

Tipo de no conformidad	Cantidad	Iteración
Redacción	4	1
Validación	2	1
Funcionalidad	1	1
Opciones que no funcionan	2	1
Excepciones	3	1
Excepciones	3	2

Tabla 17 Tipos de NC detectadas.

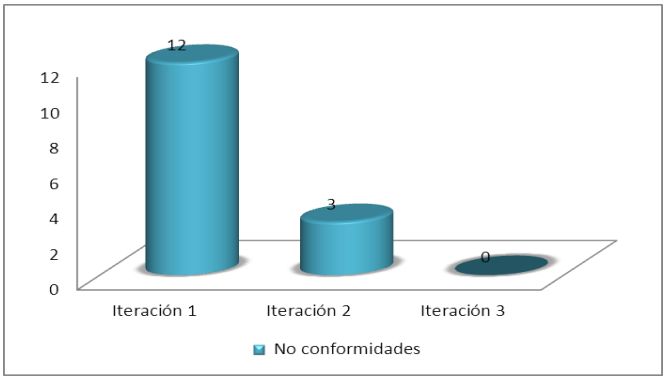

Figura 18 Resultado de las pruebas de caja negra.

3.2.3. Pruebas de Caja Blanca.

Consiste en hacer pruebas sobre los componentes individuales (subprogramas o procedimientos) de un programa.

Existen distintos tipos de pruebas:

- Pruebas unitarias.
- Pruebas de integración.
- Pruebas del sistema.
- Pruebas de implantación.
- Pruebas de aceptación.
- Pruebas de regresión.

El tipo de prueba seleccionado fue el de **pruebas unitarias**:
Las pruebas unitarias tienen como objetivo verificar la funcionalidad y la estructura de cada componente individualmente en su código. Son pruebas dirigidas a probar clases aisladamente y están relacionadas con la responsabilidad de cada clase y sus fragmentos de código más críticos. Una vez realizadas estas pruebas se asegurará que todas las sentencias del programa se han ejecutado por lo menos una vez. La técnica de camino básico fue seleccionada porque permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.
El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (Pressman, 2001)
Algunos elementos utilizados alrededor de este método de prueba son los siguientes:

- Grafo de flujo o grafo del programa: representa el flujo de control lógico de un programa y se utiliza para trazar más fácilmente los caminos de este. Cada nodo representa una o más sentencias procedimentales y cada arista representa el flujo de control.
- Complejidad ciclomática: es una métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto de las pruebas, el cálculo de la complejidad ciclomática representa el número de caminos independientes del conjunto básico de un programa. Esta medida, ofrece al probador de

software, un límite superior para el número de pruebas que debe realizar, garantizando así que se ejecutan por lo menos una vez cada sentencia.

- Camino independiente: cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición (Blanca, 2011).

Para la ejecución de las pruebas de caja blanca se seguirán los siguientes pasos descritos (Zaninotto, 2007):

Paso 1: Obtener el grafo de flujo, a partir del diseño o del código del módulo.

Paso 2: Obtener la complejidad ciclomática del grafo de flujo.

Paso 3: Definir el conjunto básico de caminos independientes.

Paso 4: Preparar un caso de prueba por camino hallado en el paso anterior.

Para mostrar la secuencia de estos se toma como ejemplo el método eliminarOficina de la clase OficinaController:

```
public void eliminarOficina()
{
    if (oficinaDAO != null) {
        if (oficina.getIdoficina() != null) {
            Oficina theOffice = oficinaDAO.findOficina(oficina.getIdoficina());
            if (theOffice != null) {
                eMS.getTransaction().begin();
                oficinaDAO.removeOficina(theOffice);
                eMS.getTransaction().commit();
            }
        }
    }

}
```

Figura 19 Método eliminarOficina de la clase OficinaController.

Paso 1: Para construir el grafo de flujo se debe introducir la notación para la representación del flujo de control, en el cual cada nodo del grafo corresponde a una o más sentencias de código. Un grafo de flujo está formado por 3 componentes fundamentales: nodos, aristas y áreas.

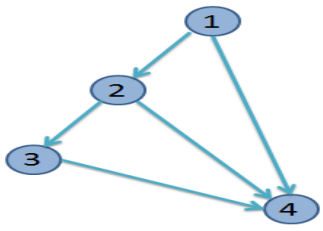

Figura 20 Grafo de flujo del método eliminarOficina de la clase OficinaController.

Paso 2: Obtener la complejidad ciclomática del grafo de flujo.

La complejidad ciclomática del código es de 3, lo que significa que existe esa cantidad posible de caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado. Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo.

Paso 3: Definir el conjunto básico de caminos independientes.

No. Camino	Camino
1	1,4
2	1,2,4
3	1,2,3,4

Tabla 18 Caminos posibles para los casos de prueba.

Paso 4: Preparar un caso de prueba por camino hallado en el paso anterior.

Descripción: Se hace la entrada de datos necesarios, validando que no se entre algún dato erróneo.

Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.

Entrada: Se muestran los parámetros que entran al procedimiento.

Resultados Esperados: Se expone resultado que se espera que devuelva el procedimiento.

Para la ejecución de las pruebas se utilizó el JUnit. Este es un entorno que permite ejecutar pruebas de clases Java, que son fáciles de hacer, leer y analizar.

Para las pruebas unitarias se realizaron dos iteraciones debido a que fueron encontrados errores en la primera. Se muestra el resultado de la prueba unitaria en la primera iteración para la clase OficinaController (ver anexo, figura 13).

En la figura 21 se muestra el resultado final de esta prueba para la clase OficinaController:

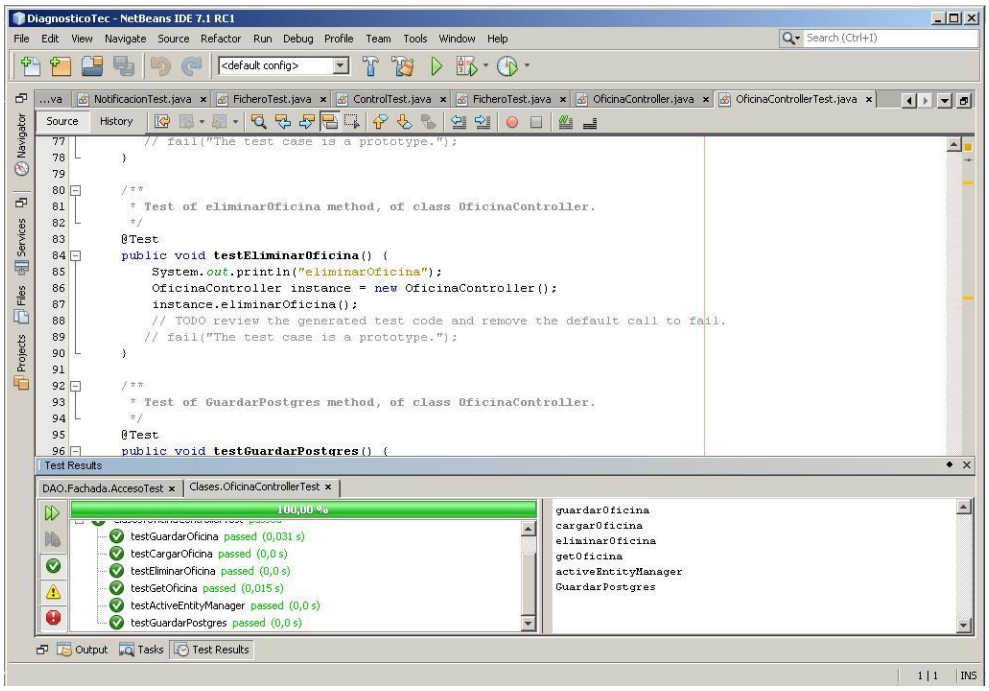

Figura 21 Iteración 2, resultado final del JUnit para la clase OficinaController

Resultado final: Como se puede apreciar para esta clase se logró alcanzar el 100% de la corrección de los métodos, así mismo se obtuvieron resultados satisfactorios para las demás clases del sistema (ver anexo, figura 14).

3.3. Conclusiones del capítulo

Los resultados obtenidos tanto en la validación del diseño como en las pruebas realizadas al sistema demuestran que el sistema está en condiciones de ser desplegado en entornos reales. Se analizaron las características de las métricas para la validación del diseño de sistemas informáticos, lo que permitió identificar las más idóneas para aplicar al diseño propuesto. Se emplearon las métricas TC, NDD y APH pertenecientes a las series LK y CK respectivamente. Como resultado de estas métricas se concluye que el diseño posee una calidad aceptable pues no es complejo, es fácil de probar y sus clases presentan baja responsabilidad y alta reutilización.

Para la validación del sistema se aplicaron pruebas de caja blanca mediante la técnica del camino básico usando el JUnit y para las pruebas de caja negra se empleó el método de partición de equivalencia; las mismas se realizaron en tres iteraciones, obteniendo resultados satisfactorios al finalizar la prueba.

CONCLUSIONES GENERALES

Con el desarrollo de este trabajo de diploma se arribó a las siguientes conclusiones:

- Los resultados del estudio de los SGI en Cuba y el mundo, demostraron la factibilidad de desarrollar un SGI para satisfacer las necesidades del proceso de diagnóstico tecnológico a entidades legales cubanas, facilitando la gestión de los datos y aportando beneficios a la toma de decisiones para el proceso de despliegue.
- La metodología ágil Scrum se consideró como la más eficaz para este tipo de proyectos de corta duración y poco personal, debido a su posibilidad de configurarse y adaptarse, con resultados satisfactorios en equipos de desarrollo.
- Fueron consideradas como herramientas, lenguajes y terminologías más apropiadas para satisfacer las necesidades del SGI propuesto las siguientes:
  - Visual Paradigm for UML 8.0 Enterprise Edition para trabajar con el lenguaje de modelado UML.
  - el lenguaje de programación Java con el IDE Netbeans 7.1 para la implementación del sistema.
  - el SGBD PostgreSQL 9.1 y SQLite 3.7.6.1 como base de datos portable para gestionar los datos.
- Se garantizó la seguridad del sistema a través de las validaciones de la entrada de datos de la entidad a la base de datos portable, la autenticación para la entrada de datos por el usuario en el centro de datos y en caso de que el fichero sqlite sea enviado mediante una red no confiable se utiliza la extensión de cifrado SQLite Encrypt que permite leer y escribir archivos de base de datos cifrados con el algoritmo AES de 256 bit.
- El Sistema de Gestión de la Información para el diagnóstico tecnológico a entidades legales cubanas fue validado a través de la aplicación de métricas de diseño y pruebas de caja negra y caja blanca, con resultados satisfactorios.

RECOMENDACIONES

Se recomienda basado en los resultados de la investigación:

- La implantación y uso del sistema en los proyectos de CEGEL para probar la eficiencia del mismo al facilitar la toma de decisiones en el proceso de diagnóstico a entidades legales cubanas.
- Aumentar los elementos contenidos en la funcionalidad encargada de graficar, en aras de visualizar más información respecto a la infraestructura de la entidad.
- En futuras implementaciones lograr generalizar la información recogida de manera que se utilice el sistema en cualquier tipo de entidad.

BIBLIOGRAFÍA

Albahari, Joseph y Albahari, Ben. 2010. C# 4.9 in a Nutshell, Fourth Edition. s.l. : O´Reilly Media, 2010. Página 1 y 2.

Arthorne, John y Laffra, Chris. 2004. Official Eclipse 3.0 Faqs. s.l. : Addison-Wesley Professiona, 2004.

Artiles Visbal, S. M. 2006. Modelo operacional para la gestión de información y conocimiento en la empresa cubana en perfeccionamiento. 2006.

BD, Oracle Berkeley. 2010. Oracle Berkeley BD. 2010.

Blanca, Caja. 2011. [En línea] 21 de diciembre de 2011. http://www.lcc.uma.es/~av/Libro/CAP1.pdf..

Böck, Heiko. 2011. The Definitive Guide to NetBeans™ Platform 7. 2011.

Booch, Grady, Rumbaugh, James y Jacobson, Ivar. 1999. Unified Modeling Language User Guide, 2de. s.l. : Addison-Wesley, 1999. pág. 482 pages.

Buschmann, F., y otros. 1996. "Pattern Oriented Software Architecture: A System of Patterns". s.l. : John Wiley & Sons, 1996.

Cámara, D. Arantzazu. 2005. Selección de Herramientas CASE. 2005.

Carreras, Julio. 2009. El Diagnóstico Tecnológico como punto de partida para la identificación de oportunidades de la empresa. 2009.

Carrillo Pérez, Isaías y Pérez González, Rodrigo. 2008. Metodología de Desarrollo de Software. 2008.

Córdova Urbano, Manuela. 2008. El Proceso de Diagnóstico y sus Elementos. 2008.

Cumming, T. , Worley, C. 2001. Organization development and change. s.l. : South Western College Publishing, 2001.

D´Souza, Desmond F. y Willis, Alan C. 1998. Objetos, componentes y Estructuras con UML, The Catalysis Aproach, Addison Wesley Longman. 1998.

de la Torre Llorente, César, y otros. 2010. Guía de Arquitectura N-Capas orientada al Dominio con .Net 4.0. . s.l. : Krasis Consulting. S.L., 2010.

Garfinkel, S., Spafford, G. y Schawartz, A. 2003. Practical Unix & Internet Security. s.l. : O'Reilly., 2003. 3rd Edition.

GECYT, Empresa de Gestión del Conocimiento y la Tecnología. 2011. [En línea] 2011. [Citado el: 25 de 12 de 2011.] http://www.gecyt.cu/old/index.php.

Gilfillan, Ian. 2003. La Biblia De Mysql. s.l. : Anaya Multimedia, 2003.

Glenford, J. Myers. 2004. The art of sotware Testing. second edition. 2004. pág. page 234.

IICA., FORAGRO. 2000. Sistema Regional de Información. 2000.

Kreibich, Jay A. 2010. Using SQLite. 2010.

Laranjeira A. , Luiz. 1990. Software Size Estimation of Object-Oriented System, IEEE Transactions on Software Engineering . 1990. Vol. 6. No. 5.

Larman, Craig. 2004. UML y PATRONES. Introducción al análisis y diseño orientado a objetos. 2004. Página 4.

Málaga, Grupo de investigación eumednet (SEJ-309) de la Universidad de. 2012. (Grupo de investigación eumednet (SEJ-309) de la Universidad de Málaga). [En línea] 2012. [Citado el: 28 de 2 de 2012.] http://www.eumed.net/libros/2010/698/Requisitos%20funcionales.html.

Martínez Martínez, Miguel Ángel. 2004. El sector asegurador como ejemplo para los sistemas de información interorganizativos: el caso del Sistema MAPFRE . 2004.

Muñoz, A. 2006. Java: del Grano a su Mesa. 2006.

NetBeans, Platform. 2012. [En línea] 2012. [Citado el: 25 de 1 de 2012.] http://netbeans.org/index_es.html.

Owens, Michael. 2006. The Definitive Guide to SQLite. 2006.

Paradigm, Visual. 2012. Visual Paradigm Online Training Center. [En línea] Visual Paradigm, 2012. [Citado el: 6 de 6 de 2012.] http://www.visual-paradigm.com/support/documents/vpumluserguide/12/13/5963_aboutvisualp.html.

PostgreSQL. 2012. PostgreSQL Global Development Group. [En línea] 2011. [Citado el: 13 de 10 de 2011.] http://www.postgresql.org/about/.

Pressman, R. S. 2001. Ingenería de Software. Un enfoque práctico. 2001.

Reynoso, Carlos Billy. 2004. Introducción a la Arquitectura de Software. Buenos Aires: s.n., 2004.

Schnicariol, Keith, Mike y Merrick. 2009. Mastering the Java™ Persistence API. 2009.

Sommerville. 2002. I.Ingeniería de Software. s.l. : Pearson Educación, 2002.

SQLite. 2012. [En línea] 2012. [Citado el: 15 de 1 de 2012.] http://www.sqlite.org/famous.html.

SQLiteEncrypt. 2011. SQLiteEncrypt. [En línea] 2011. http://www.sqlite-encrypt.com/.

Takeuchi, Hirotaka. 1999. Scrum. 1999.

Woodman, L. 1985. Information management from strategies to action. London : ASLIB, 1985. págs. p. 95-114.

Zaninotto, Fabien Potencier y François. 2007. [En línea] 21 de octubre de 2007. [Citado el: 10 de febrero de 2011.] http://www.symfony-project.org/..

Zukowski, John . 2007. Java 2 J2SE 1.4. 2007.

GLOSARIO DE TÉRMINOS

Capitalización Pascal: Estilo de codificación donde se capitaliza o se escribe en mayúscula, la primera letra de cada palabra.
Capitalización Camel: En este otro estilo se capitaliza la primera letra de cada palabra excepto la primera en palabras compuestas.

Diagnóstico: Identificación de la naturaleza o esencia de una situación o problema y de la causa posible o probable del mismo o es el análisis de la naturaleza de algo.

EclipseLink: es una completa solución de código abierto para servicios de persistencia

Garbage Collector: Es un mecanismo implícito de gestión de memoria implementado en algunos lenguajes de programación. Cuando se invoca el recolector de basura, recorre la lista de espacios reservados observando el contador de referencias de cada espacio. Si un contador ha llegado a cero significa que ese espacio de memoria ya no se usa y, por tanto, puede ser liberado.
Instituto de Ingenieros Eléctricos y Electrónicos (IEEE): Es una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación, ingenieros en informática e ingenieros en telecomunicación.
Interfaz de Programación de Aplicaciones (API): Es el conjunto de funciones o procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
Licencia Pública General de GNU o GNU General Public License: Es una licencia creada por la Free Software Foundation a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.
Proceso de diagnóstico: Razonamiento dirigido en la determinación de la naturaleza, causas u origen de un fenómeno.

Sistema de Gestión de Información: Conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades que se realizan en una organización o para automatizar los procesos de trabajo que se efectúan dentro de la misma.

Type-safe: Seguridad en la definición de tipos.

XQuery: Es un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. XQuery es un lenguaje funcional, lo que significa que, en vez de ejecutar una lista de comandos como un lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en SQL.

XPath: Es la versión de los paths para identificar archivos en una estructura de directorios. Tiene como fin el de definir un conjunto de nodos de un xml para su posterior procesamiento, se puede comparar con SQL, ya que se utiliza para recuperar información almacenada en las bases de datos relacionales.