UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Herramienta para la gestión de licencias de software para las soluciones del Centro de Informatización de la Gestión de Entidades (CEIGE)

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

Autor:

Celso Darien Montero Pérez

Tutores:

Ing. René Bauta Camejo

Ing. Dausbel Torreblanca Pavo

La Habana, Junio de 2013

"Año 55 de la Revolución"

No hay tesis que dure cien años ni tesista que los resista.

Anónimo.

Declaración de Autoría

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Celso Darien Montero Pérez **René Bauta** **Dausbel Torreblanca**

_____ _____ _____

Firma del Autor Firma de los Tutores

DATOS DE CONTACTO

Síntesis de los tutores:

Tutor: Ing. René R. Bauta Camejo

Graduado de Ingeniería en Ciencias Informáticas en Julio del 2009

Dirección: Universidad de las Ciencias Informáticas (UCI), Edificio: 129, Apto: 105

Teléfono Oficina: +53 – 7 – 8358296. Teléfono Apto: +53 – 7 – 837 – 3156

E-mail: rrbauta@uci.cu

Categoría docente: Instructor

Tutor: Ing. Dausbel Torreblanca Pavo

Graduado de Ingeniería en Ciencias Informáticas en Julio del 2012

Dirección: Universidad de las Ciencias Informáticas (UCI), Edificio: 31, Apto: 302

Teléfono Oficina: +53 – 7 – 8358292. Teléfono Apto: +53 – 7 – 835 – 8898

E-mail: dtorreblanca@uci.cu

DEDICATORIA

Dedico este trabajo a mis padres que son lo más importante en mi vida.

AGRADECIMIENTOS

A mis padres, por la confianza que siempre han depositado en mí, por los consejos y regaños oportunos que siempre me han dado, todo lo que soy se lo debo a ellos, son lo más especial que tengo en mi vida.
A mi hermana que siempre ha tratado de ser un digno ejemplo y que siempre me ha inculcado que cuando nos trazamos una meta siempre es posible alcanzarla.
A mis preciosos sobrinos que siempre van a ocupar una parte importante de mí.
A mi novia Grethel por estar siempre disponible, por ayudar tanto a mis padres cuando no he podido estar al lado de ellos, simplemente voy a estar en deuda toda la vida con ella y las palabras no serían suficiente para expresarle mi agradecimiento y afecto.
A mis tías que siempre me han tratado como a un hijo más y a mis primos que siempre se han comportado como hermanos.
A mis tutores que siempre dieron el paso al frente para cualquier inconveniente que se presentara.
A las personas que han convivido conmigo a lo largo de mi estancia en la UCI en especial agradecer al Treto, Triple, Ale, Carlos, Hector, Josue, a todos los compañeros del proyecto en especial al Crackem y Katia.
A todos los compañeros de mi grupo en especial Fernando.
A las amistades de mi municipio principalmente a Norge y a Dani que siempre han sido incondicionales conmigo.
En general a todas esas personas que me han ayudado a ser mejor cada día.

RESUMEN

Los procesos de despliegue de soluciones de software son dentro del proceso completo de desarrollo los más costosos y se ubican entre los más complicados, debido a la carga de personal necesario para esta operación, al numeroso grupo de actividades que conlleva, al soporte que se mantendrá al software, y a que este es el proceso que se realiza con la participación directa de los usuarios finales del software.

En la Universidad de las Ciencias Informáticas este proceso aumenta su complejidad debido a que esta brinda servicios de capacitación, soporte, configuración y actualización del producto. El Centro de Informatización de la Gestión de Entidades (CEIGE) como centro desarrollador de software se ha visto en la necesidad de mantener un control sobre las soluciones desplegadas para obtener mejor retroalimentación sobre estas ejecutándose en entornos reales y de esta forma brindar un mejor soporte a las mismas.

El objetivo del presente trabajo es desarrollar una herramienta que a través del otorgamiento de licencias de software permita mejorar el proceso de soporte y aumentar el control en el resto de los servicios que oferta. Para el desarrollo de la investigación se utilizaron los métodos científicos de la investigación como los métodos Análisis Histórico – Lógico y Analítico – Sintético para realizar un análisis del estado de arte profundo.

Palabras claves: Control de licencia, despliegue, herramientas, productos, software.

ÍNDICE

Índice

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Índice de tablas y figuras

INTRODUCCIÓN

La evolución de la informática como ciencia ha llevado a cambios sustanciales en la forma de pensar y actuar de la humanidad. El hombre con su insaciable sed de conocimientos se ha adentrado en un mundo en el cual la dependencia a la tecnología se hace cada vez más grande, provocando modificaciones en las esferas social, económica, política y cultural.

El software es un producto inmerso en el desarrollo tecnológico existente en la actualidad y como ocurre con otros productos creados por el hombre, está protegido por derechos de propiedad intelectual, que adquieren forma de patentes, marcas registradas o secretos industriales. Además el solo hecho de creación de un programa de ordenador confiere derechos de propiedad intelectual al informático sobre su software, sin que sea necesario llevar a cabo registro alguno.

En estos tiempos se puede conseguir cualquier tipo de software, el autor de dicho producto no puede decidir sobre sus derechos morales, pero sí tiene plena disposición sobre los patrimoniales, que son los que generan realmente beneficios económicos.

El autor de cualquier producto tiene reconocido un derecho exclusivo a explotar la obra, pudiendo decidir sobre si quiere o no conseguir beneficio económico y en qué forma. Para ello se utiliza un tipo especial de contrato denominado licencia. Para poder utilizar, modificar o distribuir un producto, ya sea mediante un convenio económico o no, tienen que adquirir esas licencias para que los usuarios ejerzan el derecho que les permite el autor. Por esto las licencias aparecen como la vía más eficiente de utilizar el software conforme a ciertas restricciones impuestas por el titular del derecho de autor y que logran la fiabilidad necesaria para los clientes.

Las licencias de software están encaminadas inicialmente a la legalidad de la distribución del producto que se comercializa. Estas particulares han traído consigo que las tendencias en la industria indiquen que la gestión del licenciamiento de software es una necesidad inmediata cuando se desea distribuir productos asociados al desarrollo de software.

Cuba se ve influenciada por cambios científicos y el uso de sistemas informáticos que permiten un mejor desempeño en el entorno empresarial y social logrando así de manera gradual empresas productoras de software. Una de las instituciones que se dedica al desarrollo de software en el país es la Universidad de las Ciencias Informáticas (UCI), dicha entidad cuenta con varios centros productivos entre los que se encuentra el Centro de Informatización de la Gestión de Entidades

(CEIGE) perteneciente a la Facultad 3. El mismo posee varios proyectos dedicados al desarrollo de productos, con el objetivo que los productos estén destinados a la gestión de entidades.
En el CEIGE existen varios productos liberados por el Centro de Calidad de Software (Calisof) y ya en proceso de despliegue como el marco de trabajo Sauxe desplegado en DESOFT[i], TRANSOFT[ii], Aduana, el sistema de seguridad Acaxia y el sistema Quarxo desplegado en el banco nacional de Cuba. Pero no se tiene un control del versionado que pueden tener las entidades que usan estos productos, además se hace imposible controlar que estas empresas distribuyan los productos del CEIGE a terceras empresas. Por lo tanto existen entidades que pudieran convertirse en mercados activos con el centro y no se gestiona de la mejor manera para que esto pase.

Actualmente el CEIGE está inmerso en una mejora continua de sus soluciones, debido a esto cuenta con varias versiones de las mismas y al no contar con un control del versionado desplegado se hace imposible saber que versión usa cada entidad con el objetivo de brindarle el soporte requerido. Asimismo puede ocurrir un error de implementación en los productos del centro ocasionando así inconsistencias en el producto, igualmente pueden presentar pérdidas de datos a gran escala que imposibiliten la recuperación de los mismos al haber adquirido el software por vías poco confiables y no tener los conocimientos necesarios para contrarrestar estos inconvenientes, en ese caso solo los fabricantes del producto podría ofrecer la asesoría necesaria que garantice la protección al cliente, puesto que el software que se adquiere ilegítimamente no es respaldado, pues se trata de una copia o imitación, no de un producto adquirido auténticamente por una persona legal o jurídica.
Conjuntamente en caso de que una empresa realmente quisiera contar con algún producto del CEIGE y adquiera una versión primitiva del mismo va a prescindir de las actualizaciones con las que se cuentan, puesto que los productos del centro se encuentran en constate evolución, estas actualizaciones se hacen necesarias porque permiten coexistencia con otros versiones del producto, integración del producto con otros de su tipo, además a medida que avanza la tecnología física o cambian los sistemas operativos, los programas se van adaptando para hacer el mejor uso posible de los recursos físicos de un equipo. Si no se tiene acceso a estas actualizaciones, se está desperdiciando la oportunidad de expandir o reforzar las herramientas de trabajo con las que se cuenten.

Asimismo puede ser que los responsables por las entidades para usar las soluciones del centro no van a contar con los conocimientos necesarios para interactuar con las mismas. Por esta pobre

noción de las entidades que pueden tener o no soluciones del CEIGE no se logra la planificación requerida de los servicios asociados a instalación, soporte, capacitación, configuración, migración de datos o actualizaciones que se pueden brindar a las empresas, por lo que se hace necesario el control de las soluciones del centro para poder gestionar eficientemente y lograr la aceptación deseada en todas las entidades que usen o deseen usar cualquier producto desplegado por el CEIGE.
A partir de la situación descrita con anterioridad, se define el siguiente **problema a resolver:** ¿Cómo aumentar el control de las soluciones desplegadas por el Centro para la Informatización de la Gestión de Entidades (CEIGE) para mejorar el proceso de soporte de las mismas?

Con el fin de darle respuesta al problema se identifica el siguiente **objetivo general:** Desarrollar una herramienta que permita aumentar el control de las soluciones desplegadas por el CEIGE a través del otorgamiento de licencias de software para mejorar el proceso de soporte de las mismas.
Para darle cumplimiento al objetivo general se propone como **objeto estudio:** proceso de otorgamiento de licencias de software, enmarcando como **campo de acción** el proceso de otorgamiento de licencias para aplicaciones web.
La investigación parte de la siguiente idea a defender **idea a defender:** Si se desarrolla una herramienta que permita aumentar el control de las soluciones desplegadas por el CEIGE a través del otorgamiento de licencias de software se podrá mejorar el proceso de soporte de las mismas.

Se plantean además, como **objetivos específicos:**

- Realizar el marco teórico de la investigación que permita identificar posibles puntos de reutilización así como los conceptos principales asociados a la gestión de licencias.
- Realizar el análisis y diseño de la propuesta de solución que sirvan de base para su posterior implementación.
- Realizar la implementación de la herramienta para la gestión del otorgamiento de licencias para los activos del CEIGE.
- Validar la propuesta de solución empleando técnicas para la validación de los requisitos, métricas para la validación del diseño y pruebas de caja blanca y caja negra para la validación de la aplicación.

Durante el desarrollo de la investigación se han utilizado un conjunto de métodos científicos los cuales se mencionan a continuación:

Métodos Teóricos:

1. **Análisis Histórico – Lógico:** Se aplica con el objetivo de conocer la existencia de herramientas encargadas de la gestión de licencias de software para aplicaciones web.
2. **Analítico – Sintético:** Se aplica con el objetivo de analizar todo lo relacionado con el licenciamiento de software y extraer información de las mismas.
3. **Métodos Empíricos: Método entrevista:** Se aplica para obtener información sobre la gestión del licenciamiento de software y las funcionalidades que debe tener la herramienta de gestión de licencias.

   Se espera alcanzar con el desarrollo de la investigación construir una herramienta que permita la gestión del control de los activos de software del CEIGE.

El presente trabajo de diploma se encuentra estructurado en cuatro capítulos que estarán guiados por los siguientes temas a tratar:

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA: Se realiza un estudio del estado del arte sobre las herramientas encargadas del licenciamiento de software, así como las herramientas y lenguajes a usar en la implementación.

CAPÍTULO II. PROPUESTA DE SOLUCIÓN: Abarca el modelo de dominio y requisitos con los que debe cumplir la solución propuesta. También se analizan los artefactos generados durante la etapa de Análisis y Diseño de la solución.

CAPÍTULO III. IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN: Se define los diagramas de componente y despliegue. Además se analizan los resultados obtenidos de la herramienta implementada y se valida la solución a través de métricas de diseño de tamaño operacional entre clases (TOC) y relaciones entre clases (REC), asimismo se realizan pruebas funcionales.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA:

El presente capítulo estará centrado en la realización de un análisis para obtener información actualizada de algunos sistemas de gestión de licencias que han sido implementados nacional e internacionalmente y conocer qué aspectos deben ser tomados en cuenta para que cualquier sistema de este tipo esté al nivel que se exige hoy en día. Se valorarán las tendencias actuales en el contexto informático partiendo de que continuamente surgen aplicaciones novedosas con mejoras que se van imponiendo en la industria del software.
El capítulo estará compuesto por conceptos generales asociados al tema de las licencias de software. Así como de un estudio de la tendencia actual del licenciamiento de software, así como las herramientas y tecnologías usadas en el desarrollo del sistema propuesto. Además de la metodología de desarrollo que será explicada a través del Modelo de Desarrollo de Software elaborado por la subdirección de producción del Centro de Informatización de la Gestión de Entidades (CEIGE) y a partir de la cual estará orientada la solución.

1.1 Conceptos Fundamentales.

- **Licencia de software:** El software es un producto de creación, en el mismo sentido que un libro, una canción o una pintura. Y como ocurre con otros productos de la creatividad humana, el software está protegido por los derechos de propiedad intelectual, que adquieren la forma de patentes, marcas registradas o secretos industriales[1].

  Es una especie de contrato, en donde se especifican todas las normas y cláusulas que rigen el uso de un determinado programa, principalmente se estipulan los alcances de uso, instalación, reproducción y copia de estos productos. Conocer las ventajas, desventajas, derechos y deberes de las empresas y de los usuarios finales, además de todas las otras personas que se relacionan con el software es imprescindible para que las empresas y los usuarios finales puedan gestionar los sistemas que serán usados[1].

  En el momento en que usted decide descargar, instalar, copiar o utilizar un determinado software, implica que usted acepta las condiciones que se estipulan en la licencia que trae ese programa, es la autorización que le permita el uso legal de determinado programa, esta licencia es un documento, ya sea electrónico, en papel original o número de serie autorizado por el autor, como obras intelectuales, no son objeto de venta, sino de cesión o licenciamiento de los derechos de la obra. Específicamente las licencias de software hacen referencia a la libertad o

falta de ella que reciben los usuarios de un software desarrollado por parte de su(s) autor(es) para poder usarlo y/o modificarlo de acuerdo a unas condiciones previamente pactadas[1].

- **Licenciante:** El licenciante o proveedor-licenciante es aquel que provee el software más la licencia al licenciatario, la cual, le permitirá a este último tener ciertos derechos sobre el software. El rol de licenciante lo puede ejercer cualquiera de los siguientes actores: [2]
- **Autor:** El desarrollador o conjunto de desarrolladores que crea el software son por antonomasia quienes en una primera instancia poseen el rol de licenciante al ser los titulares originales del software.

- **Titular de los derechos de explotación**: Es la persona natural o jurídica que recibe una cesión de los derechos de explotación de forma exclusiva del software desde un tercero, transformándolo en titular derivado y licenciante del software.
- **Distribuidor**: Es la persona jurídica a la cual se le otorga el derecho de distribución y la posibilidad de generar sublicencias del software mediante la firma de un contrato de distribución con el titular de los derechos de explotación.
- **Licenciatario:** El licenciatario o usuario licenciatario es aquella persona física o jurídica que se le permite ejercer el derecho de uso más algún otro derecho de explotación sobre un determinado software cumpliendo las condiciones establecidas por la licencia otorgada por el licenciante[2].

- **Usuario consumidor:** Persona natural que recibe una licencia de software otorgada por el licenciante, la cual, se encuentra en una posición desventajosa ante los términos y condiciones establecidas en ella.
- **Usuario profesional o empresa:** Persona natural o jurídica que recibe una licencia de software otorgada por el licenciante, la cual, se encuentra en igualdad de condiciones ante el licenciante para ejercer sus derechos y deberes ante los términos y condiciones establecidos en la licencia.
- **Elementos objetivos de una licencia de software:**

- ✓ **Plazo**: El plazo determina la duración en el tiempo durante la cual se mantienen vigentes los términos y condiciones establecidos en licencia. Las licencias en base a sus plazos se pueden clasificar en, licencias con plazo específico, indefinido y sin especificación de plazo[2]:
- ✓ **Precio:** El precio determina el valor el cual debe ser pagado por el licenciatario al licenciante por el concepto de la cesión de derechos establecidos en la licencia[2].

- **Garantía de titularidad:**

Es la garantía ofrecida por el licenciante o propietario, en la cual, asegura que cuenta con suficientes derechos de explotación sobre el software como para permitirle proveer una licencia al licenciatario[2] .

1.2 Tendencias actuales en el licenciamiento del software.

Además de proteger el software y la propiedad intelectual, se necesita proteger los ingresos de las ventas de los productos. Para el aseguramiento de que el software sólo está disponible para los usuarios adecuados, de acuerdo con los términos que se definan, este proceso se controla a través de las licencias. Las licencias proporcionan la flexibilidad necesaria para implementar las estrategias del negocio para la distribución del software.

Para obtener el máximo beneficio de la estrategia de licencias en una empresa, es necesario un sistema de licencias de software que le proporcione la flexibilidad para adaptar los términos de licencias, de modo que coincidan con la estrategia de negocio y para adaptarse rápidamente a los cambios del mercado y a las necesidades de la empresa.

- **Soluciones basadas en hardware.**

Las soluciones basadas en hardware, proporcionan un dispositivo hardware externo junto con el software. El funcionamiento de su software dependerá de la conexión del dispositivo al ordenador del usuario final. En el tiempo de ejecución, el software se comunica con el dispositivo hardware y sólo funciona correctamente si recibe una respuesta verosímil de éste[3].

- **Soluciones basadas en software.**

Con las soluciones basadas en software, basta con seguir la instalación del software del equipo del usuario final para que la protección y licencias se asocien a ese equipo en concreto. El software sólo funcionará cuando el usuario haya introducido la llave del producto. En el tiempo de ejecución, el servidor comprueba que el software está en el equipo que tiene licencia para ejecutarlo y que se está utilizando de acuerdo con los términos de licencia del usuario[3].

Se decide usar partes de ambas soluciones, puesto que para generar las licencias hace falta datos del hardware de la máquina pero el peso de la solución estará basado en software.

1.3 Herramientas de otorgamiento de licencias.

En el mundo existen una gran cantidad de herramientas encargadas de la gestión del otorgamiento de licencias de software. Se escogen las siguientes herramientas puesto que son soluciones que pueden aportar ideas significativas a la hora de elaborar una herramienta para el CEIGE.

- ***NetSupport* DNA:** Es una completa solución que permite realizar inventarios de hardware y de software así como la gestión de licencias. Permite asignar información de licencias a los datos activos de una PC. Tiene una interfaz muy amigable que permite que los operadores puedan asignar rápidamente licencias a varios ordenadores. Además permite monitorear el uso de licencias dentro de una organización. Al introducir el número de licencias para cada aplicación es posible identificar el uso de ellas cada vez que se ejecute el inventario de software[4].

- **MSIA (Analizador de Inventario):** Explora las estaciones de trabajo individuales o redes completas de PC para los productos de software de Microsoft. MSIA genera los informes que indican los títulos de software, tipo y número de licencias para cada título, el número de instalaciones para ese título y los vínculos a una lista de ordenadores explorados. El MSIA puede funcionar con redes que cuentan con hasta 250 o menos ordenadores[1].

- ***ElecKey*:** Es la solución completa para la protección de copia de software, licencias de software y distribución de software de forma segura. *ElecKey* proporciona la capacidad de concesión de licencias de software , permitiendo así crear una gran variedad de modelos de licencia, por ejemplo, por un plazo limitado versión de evaluación , licencia flotante, la activación del software, entre otros[5].

  Una vez que se distribuye el software, *ElecKey* ofrece una amplia variedad de opciones que permiten gestionar la licencia y la concesión de licencias de un servidor de activación completamente automatizado[5].

  Por otra parte, también proporciona la solución para hacer frente a las difíciles cuestiones de soporte al usuario final de concesión de licencias. Puede crear las utilidades de usuario final, junto con la aplicación protegida que permitirá al usuario activar, desactivar, transferir y retirar la licencia de software fácil y cómoda. Mediante estos servicios se puede verificar si el usuario ha destruido la licencia antes de la emisión de una restitución[5].

- **Protector de Licencias:** Administra y controla licencias de software, genera versiones de prueba, proporciona un programa de protección contra copia y soporta pruebas de usuario concurrentes. Posibilita generar licencias automáticamente para tiendas en línea. Ofrece archivos de licencia cifrados con claves hechas por el cliente para cada proyecto de informática y Claves de Activación Segura que pueden ser usadas sólo una vez[6].

- ***PACSViewer*:** Es una herramienta desarrollado en la UCI para facilitar el proceso de gestión del otorgamiento de licencias para al Departamento de Software Médico Imagenológico, logrando así la activación del sistema alas *PACSViewer* de forma más eficiente y segura, esta herramienta se encarga de otorgar licencias de software pero únicamente a un solo producto[7].

- **Valoración de las herramientas encargadas de la gestión del otorgamiento de licencias de software:**

Debido a que algunos de estos sistemas son propietarios y por ende sus costos para uso son elevados, financiar los gastos de la implantación de un sistema de los existentes en el mundo, para la gestión del otorgamiento de licencias de software sería muy difícil para países subdesarrollados como Cuba.

Todo esto ligado a que estos sistemas no cubren en su totalidad con lo que se quiere diseñar, puesto que algunas monitorean el uso de licencias pero no logran asignarlas, se limitan además a chequear la cantidad de instalaciones de cada sistema informático, con el objetivo de verificar el cumplimiento de las licencias, así como presentan limitaciones en cuanto a la durabilidad del licenciamiento y no gestionan servicios asociados a instalación, soporte, actualización, configuración etc.

Además algunos de estos sistemas funcionan sólo para productos de empresas específicas, por lo que el Centro de Informatización de la Gestión de Entidades (CEIGE) no tendría acceso a las mismas.

Es por ello que se decide desarrollar una herramienta para facilitar el proceso de gestión del otorgamiento de licencias para el CEIGE, aunque con el estudio de las herramientas se tomarán en cuenta particularidades de algunas de ellas que se puedan incorporar a la solución propuesta.

1.4 Metodología de Desarrollo.

La selección de una metodología para el desarrollo de un determinado sistema está condicionada por varios factores que definen la más apropiada, entre ellos se encuentran, el tipo de sistema a desarrollar, el personal participante, los recursos disponibles y las necesidades del cliente, por lo que no existe una

metodología estándar a seguir, más bien se necesita que este proceso sea adaptable y configurable de acuerdo al proyecto. Existen dos tipos fundamentales de metodologías para tratar este tema, las que siguen los métodos tradicionales, que son generalmente para software de gran envergadura, y las que proponen procesos ágiles para el desarrollo, más usadas en pequeños software y con marcadas diferencias entre sí[8].

Todo el proceso relacionado con el desarrollo de la solución propuesta será regido por el "Modelo de desarrollo de Software" definido por la subdirección de producción de CEIGE en su versión 1.0 actualizada el 29/10/2012. Dicho modelo en resumen, plantea lo siguiente:

El proceso de informatización de la sociedad cubana, ha propiciado el aumento del uso de herramientas informáticas en los principales sectores del país. En tal sentido, la Universidad de las Ciencias Informáticas (UCI) desempeña un rol protagónico desde su surgimiento mediante la producción de soluciones y servicios informáticos. Adscrito a la Facultad 3 de la propia universidad se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE). Dicho centro productivo dirige sus resultados hacia la esfera de la gestión de entidades. La producción se concentra en el desarrollo de proyectos generalmente de gran magnitud, por lo que se hace necesario contar con un modelo estandarizado, que establezca las distintas fases por las que se debe transitar y el conjunto de artefactos a generar en cada una de ellas. Teniendo como precedente dicha necesidad y en colaboración con las distintas líneas de desarrollo donde se ejecutan cada uno de estos proyectos, el presente documento propone el Modelo de desarrollo de software para el CEIGE[8].

Características:

- ✓ Iterativo e Incremental
- ✓ Orientado a componentes
- ✓ Ágil y adaptable al cambio
- ✓ Centrando en la arquitectura

1.5 Tecnologías y herramientas para el desarrollo.

1.5.1 Herramientas CASE

Conjunto de programas que asisten a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un Software.

Las herramientas CASE [iii] proporcionan al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería. Al igual que las herramientas de ingeniería y de diseño asistidos por computadora que utilizan los ingenieros de otras disciplinas. Las herramientas CASE ayudan a asegurar la calidad de un producto desde su diseño antes de construirlo.

- **Visual Paradigm**: es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software. Ayuda a una construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Permite la generación de bases de datos, la transformación de diagramas de entidad relación en tablas de base de datos. También proporciona una mayor documentación de la Base de Datos y diagramas de mapeo de relación de objetos. Esta herramienta funciona sobre múltiples plataformas, es un producto de muy buena calidad, así como las imágenes y reportes generados por la misma. Además soporta aplicaciones web, es fácil de instalar y actualizar y es compatible con otras ediciones del producto[9].

1.5.2 Herramientas para el desarrollo colaborativo

- ***Subversion*:** Es un sistema de control de versiones libre (código abierto). Es decir, *Subversion* maneja ficheros y directorios a través del tiempo. Un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar la historia de cómo cambiaron sus datos. En este sentido, muchas personas piensan en un sistema de control de versiones como una especie de "máquina del tiempo"[10].

  Subversion puede acceder a su repositorio a través de la red, lo que permite que pueda ser utilizado por personas en diferentes equipos. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto a través del cual todas las modificaciones que se producen. Y debido a que el trabajo se versiona, no tienen por qué temer que la calidad es la compensación por la pérdida de ese conducto, si algún cambio incorrecto que se haga a los datos, simplemente deshaga ese cambio[10].
- **RapidSVN:** es una multiplataforma para el sistema de revisión de *Subversion* escrito en C++ utilizando el marco wxWidgets. Este proyecto también incluye un cliente de *Subversion* C++ API.

Con *Subversion* se quiere construir un mejor cliente visual que utilice las mejores características de los clientes de otras arquitecturas de control de revisiones. Si bien es bastante fácil para los nuevos usuarios de *Subversion* para trabajar con él también debe ser lo suficientemente potente como para hacer que los usuarios experimentados aún más productivos[11].

1.5.3 Herramientas para el desarrollo

- **PostgreSQL:** Lleva varios años de desarrollo activo y una arquitectura probada, ha ganado una reputación fuerte para la fiabilidad, integridad de los datos y exactitud. Corre en todos los sistemas operativos incluidos Linux y Windows. También apoya el almacenamiento de grandes objetos binarios, incluso imágenes, sonidos y videos. Tiene interfaces nativas de la programación para los lenguajes C++, Java.net, Perl, Python, Ruby, entre otros y una documentación excepcional. Entre sus principales características pueden encontrarse[12]:

- ✓ Disponible totalmente sin costo alguno.
- ✓ Disponible para los Sistemas Operativos UNIX y Windows.
- ✓ Soporte total del Modelo Relacional de Bases de datos.
- ✓ Extensiones propias a SQL para realizar consultas sobre la base de datos.
- ✓ Dependencias entre objetos, integridad referencial.
- ✓ Soporta valores no atómicos como dominio de un campo.

Sistema de gestión de BD[iv] relacional, incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. Es un poderoso manejador de base de datos de código abierto diseñado para administrar grandes volúmenes de datos[12]. Se estará haciendo uso de PostgreSQL en su versión 8.3 u superior

- **Netbeans:** Es un entorno de desarrollo integrado (IDE) disponible para Windows, Mac, Linux y Solaris pensado para escribir, compilar, depurar y ejecutar programas. Consiste en un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones web, empresariales, de escritorio y aplicaciones móviles utilizando la plataforma Java, así como PHP, JavaScript y Ajax, entre otros. Existe además un número importante de módulos para extender el IDE NetBeans. El IDE NetBeans es un producto libre y gratuito sin restricciones de uso[13]. Se estará haciendo uso en su versión 7.0 o superior

1.5.4 Librerías y marcos de trabajo

- **Sauxe:** Es un marco de trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. Sauxe da solución a un sin número de escenarios o aspectos arquitectónicos como: gestión y configuración dinámica de cache, integración de componentes de forma distribuida o no distribuida, acceso a bases de datos a través de una capa de abstracción, gestión de concurrencia de recursos, administración centralizada de transacciones, gestión dinámica de las trazas generadas por los sistemas, implementación de mecanismos de autenticación y autorización, implementación de mecanismos de mensajería y control de excepciones, gestión y configuración dinámica de precondiciones, pos condiciones y validaciones de variables, gestión y configuración de flujos de trabajo, visualización de las funcionalidades de un sistema, entre otros escenarios de alta complejidad, garantizando los atributos de calidad de los sistemas que se desarrollen con el mismo[14].

- **ExtJS:** es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM. Dispone de un conjunto de componentes para incluir dentro de una aplicación web, como[15]:

- ✓ Cuadros y áreas de texto.
- ✓ Campos para fechas.
- ✓ Campos numéricos.
- ✓ Combos.
- ✓ Radio buttons y check boxes.
- ✓ Editor HTML.
- ✓ Árbol de datos.
- ✓ Pestañas.
- ✓ Barra de herramientas.
- ✓ Menús
- ✓ Paneles divisibles en secciones.
- ✓ Sliders.
- ✓ Gráficos

Además la ventana flotante que provee ExtJS es excelente por la forma en la que funciona. Al moverla o redimensionarla solo se dibujan los bordes haciendo que el movimiento sea fluido lo cual le da una ventaja tremenda frente a otros[15].

- **Beneficios:**

- ✓ Existe un balance entre Cliente–Servidor. La carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.
- ✓ Comunicación asíncrona.
- ✓ Eficiencia de la red. El tráfico de red puede disminuir al permitir que la aplicación elija que información desea transmitir al servidor y viceversa, sin embargo la aplicación que haga uso de la pre-carga de datos puede que revierta este beneficio por el incremento del tráfico.

Una de las grandes ventajas de utilizar ExtJS es que permite crear aplicaciones complejas utilizando componentes predefinidos así como un manejador de diseños similar al que provee Java Swing[v], gracias a esto provee una experiencia consistente sobre cualquier navegador, evitando el tedioso problema de validar que el código escrito funcione bien en cada uno. Además, la ventana flotante que provee ExtJS es excelente por la forma en la que funciona. Al moverla o redimensionarla sólo se dibujan los bordes haciendo que el movimiento sea fluido lo cual le da una ventaja tremenda frente a otros[15].

- **Zend Framework:** Es una librería de código abierto de componentes escritos en PHP5, orientada a objetos (OO), que facilita el desarrollo de sitios web. Zend Framework hace hincapié en la calidad del código, a través de una batería de test unitarios, utilizando PHPUnit[16].

Características.

- ✓ Trabaja con MVC (Modelo Vista Controlador)
- ✓ Cuenta con módulos para manejar archivos PDF, canales RSS, Web Services.
- ✓ El Marco de Zend también incluye objetos de las diferentes bases de datos, por lo que es extremadamente simple para consultar base de datos, sin tener que escribir ninguna consulta SQL[16].
- ✓ Una solución para el acceso a base de datos que balancea el ORM con eficiencia y simplicidad.
- ✓ Completa documentación y pruebas de alta calidad.

- ✓ Soporte avanzado.
- ✓ Robustas clases para autenticación y filtrado de entrada.
- ✓ Muchas otras clases útiles para hacerlo tan productivo como sea posible[16].

- **Doctrine ORM:** es un potente y completo sistema ORM (Mapeador de Objetos Relacionales) para PHP 5.2+ con un DBAL (Base de Datos de Capa de Abstracción) incorporado. Se está empezando a ver su potencial, pero de la documentación se puede decir que tiene todas las características necesarias para ser funcional en casi cualquier proyecto. Entre muchas otras cosas brinda la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir convertir clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos. Por otro lado, como la librería es bastante grande ésta tiene un método para ser compilada al pasar a producción[17].

  Cuando se trabaja con Doctrine, se necesita informar a su motor interno de cuál es el modelo de la aplicación, para ello se puede hacer ingeniería inversa de la base de datos existente, o si se empieza la aplicación desde 0, crear el modelo en la sintaxis específica que nos propone Doctrine y luego generar toda la base de datos. Para crear el modelo, doctrine brinda dos alternativas, hacer una clase por tabla e indicarle mediante PHP el tipo de datos que almacenaremos en él.

1.5.5 Lenguajes de modelado y desarrollo:

Lenguaje de modelado

- **UML:** Se utilizará el lenguaje de modelado UML como soporte para el modelo de la BD. Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Probablemente, una de las innovaciones conceptuales en el mundo tecnológico del desarrollo de software que más expectativas y entusiasmos haya generado en muchos años[18].

  UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables[18]. Se estará utilizando la versión 5.0 o superior.

Lenguajes de desarrollo

- **PHP:** es un lenguaje de scripting de propósito general ampliamente utilizado que es especialmente adecuado para el desarrollo web y puede ser embebido en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl. Su meta es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil[19].
- **JavaScript:** es el lenguaje interpretado orientado a objetos desarrollado por Netscape que se utiliza en millones de páginas web y aplicaciones de servidor en todo el mundo. JavaScript de Netscape es un conjunto del lenguaje de scripts estándar que presenta sólo leves diferencias respecto a la norma publicada[20].

  JavaScript es un lenguaje de programación dinámico que soporta construcción de objetos basado en prototipos. La sintaxis básica es similar a Java y C++ con la intención de reducir el número de nuevos conceptos necesarios para aprender el lenguaje. Las construcciones del lenguaje funcionan de manera similar que en estos lenguajes[20].

  JavaScript puede funcionar como lenguaje procedimental y como lenguaje orientado a objetos. Los objetos se crean programáticamente añadiendo métodos y propiedades a lo que de otra forma serían objetos vacíos en tiempo de ejecución, en contraposición a las definiciones sintácticas de clases comunes en los lenguajes compilados como C++ y Java. Una vez se ha construido un objeto, puede usarse como modelo (o prototipo) para crear objetos similares[20].

1.6 Patrones

Muchos de los escritores que emprenden el tema de los patrones hacen referencia a la obra del arquitecto Christopher Alexander, quien presentó por primera vez un lenguaje de patrones para poder diseñar ciudades, barrios, grupos de edificios, plazas, edificios, habitaciones.

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, además describe el núcleo de la solución de ese problema de forma que se puede utilizar esta solución un millón de veces, sin hacerlo nunca de la misma manera [21].

1.6.1 Patrones de diseño

Los patrones del diseño abordan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes. Existen diversas formas de implementar patrones de diseño.

Un patrón de diseño es una abstracción de una solución en un alto nivel. Los patrones dan solución a problemas existentes en muchos niveles de abstracción. Hay patrones que comprenden las diferentes etapas del desarrollo; desde el análisis hasta el diseño y desde la arquitectura hasta la implementación[22].

1.6.2 Patrones para la asignación de responsabilidades (GRASP)

Los patrones GRASP representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. GRASP es el acrónimo para (Patrones Generales de Software para Asignar Responsabilidades)[23].

- ✓ **Experto:** Se encarga de asignar una responsabilidad al experto en información, o sea, aquella clase que cuenta con la información necesaria para cumplir la responsabilidad.
- ✓ **Creador:** Este patrón es el responsable de asignarle a la clase B la responsabilidad de crear una instancia de clase A. B es un creador de los objetos A.
- ✓ **Alta Cohesión:** Asigna una responsabilidad de forma tal que la cohesión siga siendo alta.
- ✓ **Bajo Acoplamiento:** Este patrón es el encargado de asignar una responsabilidad para conservar bajo acoplamiento.
- ✓ **Controlador:** Asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase.

1.6.3 Patrón *Gang of Four* (GOF)

Los patrones GoF ((***Gang of Four***) Pandilla de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento [23].

- ✓ ***Singlenton* o Solitario:** El patrón *Singleton* es patrón creacional que asegura que exista una única instancia de una clase. A primera vista, uno puede pensar que pueden utilizarse clases con miembros estáticos para el mismo fin. Sin embargo, los resultados no son los mismos, ya que en este caso la responsabilidad de tener una única instancia recae en el cliente de la clase. El patrón *Singleton* hace que la clase sea responsable de su única instancia, quitando así este problema a los clientes[24].

1.7 Conclusiones del Capítulo

Durante este capítulo se consolidó el basamento teórico con vistas al desarrollo del software mediante un estudio del estado del arte de las principales herramientas que son usadas para la gestión del

licenciamiento de software obteniendo como resultado del estudio que ninguna de las herramientas estudiadas cumple con las necesidades del CEIGE por lo que se decide desarrollar una herramienta que gestione el control de los activos de software del centro.

Conjuntamente se realiza estudio de las herramientas y lenguajes que se pueden usar definiendo así el uso del marco de trabajo Sauxe, las librerías ExtJS, Zend Framework, Doctrine, así como lenguaje de modelado el UML y de desarrollo PHP y Java Script.

Se define el modelo de desarrollo a seguir, el cual describe cada artefacto a obtener en cada etapa del desarrollo de esta aplicación y así se sientan las bases para su inicio.

CAPÍTULO II. PROPUESTA DE SOLUCIÓN:

2.1 Introducción:

En éste capítulo se realiza una propuesta de solución para la gestión del control de licencias de software de los productos del CEIGE. Dicha propuesta va a estar conformada por la identificación y descripción de los requisitos funcionales y no funcionales y los artefactos generados durante las fases de Modelado del Negocio, Requisitos y Análisis y Diseño.

2.2 Propuesta de Solución:

Con la herramienta para la gestión del control de licencias de software de los productos del Centro de Gestión de Entidades se pretende controlar de forma más estricta la distribución de los productos desarrollados por el centro.

2.3 Modelo Conceptual:

Un modelo conceptual es una representación de conceptos en un dominio del problema. Muestra asociaciones entre conceptos y atributos, además de descomponer el espacio del problema en unidades comprensibles. La creación de este modelo contribuye a esclarecer la terminología o nomenclatura del dominio. Se puede ver como un modelo que comunica cuales son los términos importantes y como se relacionan entre sí[25].

El modelo de desarrollo del CEIGE en la disciplina de modelado de negocio define un modelo de dominio en el que se elabora un modelo conceptual[8].

Después de haberse realizado entrevistas a especialistas del departamento de tecnología del CEIGE y a especialistas de ALBET [vi]y el Centro de Soporte [vii]se definen los conceptos asociados a la solución los cuales se recogen en el siguiente modelo conceptual.

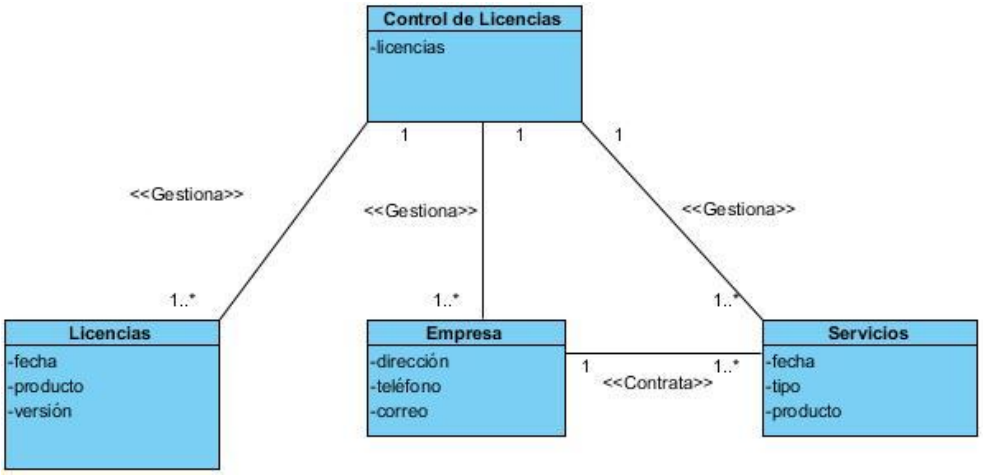


Figura 1: Modelo Conceptual

2.3 Requisitos de Software:

El propósito de este acápite es especificar las condiciones o capacidades que el sistema debe cumplir y las restricciones bajo las cuales debe operar, logrando un entendimiento entre el equipo de desarrollo y el cliente, y especificando las necesidades reales de forma que satisfaga sus expectativas. Una vez definidos claramente los requisitos funcionales y no funcionales que va a tener el sistema, se debe realizar una descripción detallada de cada uno de ellos, de forma que sea entendible por los clientes y usuarios[26].

Para la captura de requisitos se utilizaron las técnicas descritas a continuación:

- ✓ **Tormenta de ideas:** No es más que reuniones en grupos cuyo objetivo es que cada participante muestre su idea libremente. Para esto se realizaron talleres con los integrantes del equipo de desarrollo tanto como los tutores con el objetivo de debatir del tema y así definir las principales funcionalidades[26].
- ✓ **Entrevista con el cliente**: Se realizan reuniones con el/los cliente/s para conocer sus exigencias con mayor claridad y sentar las bases para la captura de los requisitos[26].
- ✓ **Observación de sistemas semejantes**: Se realiza un estudio de herramientas que sus soluciones se asemejen a la propuesta de solución[26].

2.3.1 Requisitos Funcionales del Sistema:

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir una vez estudiado el problema y los posibles procesos a informatizar, de forma tal que se defina cómo el sistema o producto final se ajustará mejor a las necesidades del negocio y cómo va a ser utilizado éste por los usuarios[26].

A partir de los artefactos obtenidos y teniendo en cuenta las técnicas empleadas para la captura de requisitos (**Tormenta de ideas, Entrevista con el cliente, Observación de sistemas semejantes**) se identifican cuatro agrupaciones funcionales que se desglosan de la siguiente manera:

- ✓ Gestionar licencias.
- ✓ Gestionar servicios.
- ✓ Gestionar nomservicios.
- ✓ Gestionar datos de la empresa.

A continuación se muestra la descripción asociada al requisito funcional Gestionar licencias, las otras descripciones se encuentran en el expediente de proyecto de SAUXE.

RF1: Adicionar Licencias:

Precondiciones	El usuario se ha autenticado ante el sistema y tiene permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico Adicionar Licencias.	
1	Se selecciona al sistema al cual se le va a adicionar una licencia.
2	Una vez seleccionado el sistema se escoge la opción adicionar.
3	El usuario introduce los datos a adicionar: Núm. de activación: Emitida por:
4	El sistema valida los datos introducidos.
5	Si los datos son correctos el sistema crea la licencia.
6	Concluye el requisito.
Pos-condiciones	
1	Se crea una nueva licencia.
Flujos alternativos	

Flujo alternativo 4.a Datos erróneos	
1	El sistema señala los datos incorrectos y permite insertar los datos correctos nuevamente.
2	El usuario corrige los datos.
3	Volver al paso 3 del flujo básico.
Pos-condiciones	
1	N/A.
Flujo alternativo 4.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso número 3 del flujo básico.
Pos-condiciones	
1	N/A.
Flujo alternativo 2.c El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
2	No se registran los datos.
Validaciones	
1	El sistema valida los datos introducidos.
Conceptos	**Licencias** Visibles en la interfaz: Adicionar Modificar Eliminar Descargar
Requisitos especiales	N/A
Asuntos pendientes	N/A

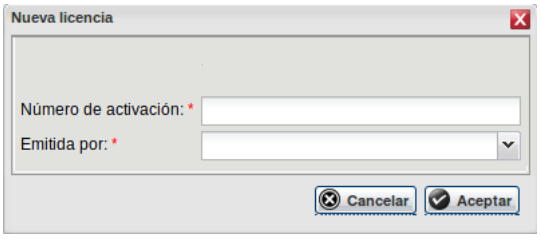


Figura 2: Interfaz gráfica de usuario Nueva licencia

RF2: Modificar Licencias:

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico Modificar licencia	
1	Se selecciona la licencia y luego se selecciona la opción modificar
2	El sistema valida los datos introducidos.
3	Si los datos son correctos el sistema los registra.
4	Concluye el requisito.
Pos-condiciones	
1	Se realizó la modificación de la licencia seleccionada
Flujos alternativos	
Flujo alternativo 3.a Datos erróneos	
1	El sistema señala los datos incorrectos y permite insertar los datos correctos nuevamente.
2	El usuario corrige los datos.
3	Volver al paso 3 del flujo básico.
Pos-condiciones	
3	N/A
Flujo alternativo 3.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso número 3 del flujo básico.
Pos-condiciones	
1	N/A.
Flujo alternativo 1.c El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se registran los datos.
Validaciones	
1	El sistema valida que los datos introducidos estén correctos.
Conceptos	**Gestionar Licencias** Visibles en la interfaz: Adicionar Modificar Eliminar Descargar

Requisitos especiales	N/A
Asuntos pendientes	N/A.

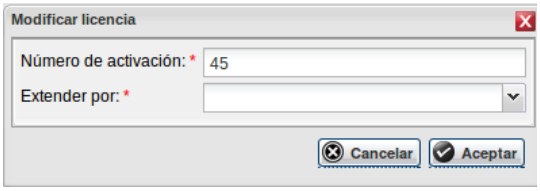


Figura 3: Interfaz gráfica de usuario Modificar licencia

RF3: Eliminar Licencias:

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción. Que la licencia que desea eliminar, se encuentre cargada en la aplicación.
Flujo de eventos	
Flujo básico Eliminar licencia	
1	Se selecciona la licencia que se va a eliminar.
2	El sistema elimina la licencia seleccionada.
3	Concluye el requisito.
Pos-condiciones	
Se elimina la licencia seleccionada.	
Flujos alternativos	
Flujo alternativos 3*.a No se elimina la Licencia	
1	Volver al paso 2 del flujo básico
Flujo alternativo*.b El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se registran los datos.
Validaciones	

1	N/A.	
Conceptos	**Licencias**	Visibles en la interfaz: Adicionar Modificar Eliminar Descargar
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

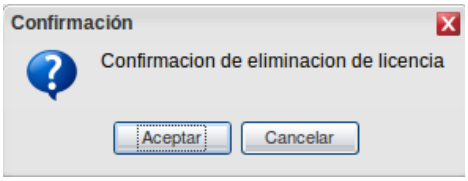


Figura 4: Interfaz gráfica de usuario Confirmación

RF4: Descargar Licencias:

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción. Que la licencia que desea descargar, se encuentre cargada en la aplicación.
Flujo de eventos	
Flujo básico Descargar licencia	
4	Se ejecuta la acción descargar una vez seleccionada la licencia.
5	El sistema acepta la acción y descarga la licencia seleccionada.
6	Concluye el requisito.
Pos-condiciones	
Se descarga la licencia seleccionada.	
Flujos alternativos	
Flujo alternativos 3*.a No se descarga la Licencia	
2	Volver al paso 2 del flujo básico
Flujo alternativo 2*.b El usuario cancela la acción	

2	Concluye el requisito.
Pos-condiciones	
2	No se registran los datos.
Validaciones	
2	N/A.

Conceptos	**Licencias**	Visibles en la interfaz: Adicionar Modificar Eliminar Descargar
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

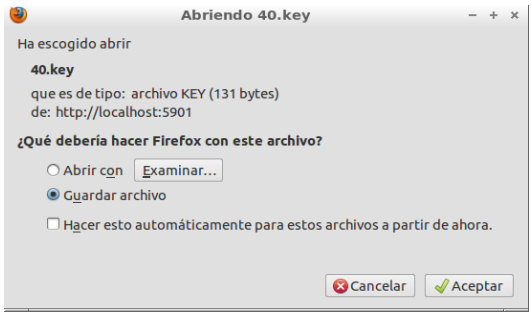


Figura 5: Interfaz gráfica de usuario *Guardar llave*

2.3.2 Requisitos no Funcionales:

Los requisitos no funcionales son propiedades o cualidades que el sistema debe cumplir, éstos ni describen información a guardar, ni funciones a realizar, a través de los mismos se debe obtener un producto atractivo, usable, rápido y confiable.[26]

- **Usabilidad:**

✓ **RnF 1**: El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

- **Eficiencia:**

✓ **RnF 2**: Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, no mayores de 5 segundos para las actualizaciones y 20 para las recuperaciones.

- **Soporte:**

✓ **RnF 3**: La aplicación contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración.

✓ **RnF 4:** Los desarrolladores deben utilizar para el desarrollo de la solución el estándar de codificación elaborado en el Departamento de Tecnología[27].

- **Restricciones de diseño:**

✓ **RnF 5:** El lenguaje de programación a utilizar para desarrollar el sistema debe ser PHP5.3.10 para la lógica del negocio y ExtJS2.2 para la capa de presentación.

✓ **RnF 6:** Las herramientas a utilizar para desarrollar el sistema deben ser PostgreSQL8.3 como gestor de base de datos y Pgadmin como cliente, Doctrine1.2.2 para la capa de acceso a datos y ZendExt Framework1.11 para la lógica del negocio.

✓ **RnF 7:** El sistema debe ser multiplataforma, haciendo énfasis en Linux y Windows.

- **Ambiente:**

Para que el sistema funcione es necesario garantizar los siguientes requisitos de software:

- **Para el cliente:**

✓ **RnF 8**: Navegador Mozilla Firefox. v3.6 o superior.

✓ **RnF 9:** Sistema operativo XP o superior o Linux en cualquiera de sus distribuciones.

- **Para el servidor:**

✓ **RnF 10:** Sistema operativo XP o superior o en Linux en cualquiera de sus distribuciones.

✓ **RnF 11:** Un servidor Apache v2.0 o superior con módulo PHP v5.0 o superior que esté disponible, además este debe estar configurado con la extensión “pgsql” incluida.

✓ **RnF 12**: Un servidor de base de datos PostgreSQL v8.3 o superior.

Para que el sistema funcione es necesario garantizar los siguientes requisitos de hardware:

- **Para el servidor**:

- ✓ **RnF 13:** Requerimientos mínimos: Procesador Pentium III a 1GHz de velocidad de procesamiento y 1Gb de memoria RAM.
- ✓ **RnF 14:** Al menos 40Gb de espacio libre en disco duro.
- ✓ **RnF 15:** Tarjeta de red.

- ➢ **Para el cliente:**
- ✓ **RnF 16**: Requerimientos mínimos: Procesador Pentium II a 133Mhz con 128 Mb de memoria RAM.
- ✓ **RnF 17:** Tarjeta de red.

2.3.3 Técnicas de validación de requisitos:

La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema que han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos. Una especificación es considerada con calidad por el estándar IEEE 830[viii] cuando es correcta, no ambigua, completa, consistente, ordenada por importancia y estabilidad, verificable, modificable y trazable.

Muchas son las técnicas para la validación de los requisitos, entre ellas se puede mencionar: auditorías, matrices de trazabilidad, generación de casos de pruebas, análisis de consistencia automático (CASE, BD requerimientos), revisión técnica formal y el prototipo. Para validar los requisitos propuestos se utilizaron las técnicas[28]:

- ✓ **Revisión técnica formal:** Es un proceso manual que involucra a ambas partes, tanto cliente como equipo de desarrollo, en la revisión formal el equipo de desarrollo conduce al cliente a través de los requerimientos, explicándole las implicaciones de cada uno. Los conflictos, contradicciones, errores y omisiones deben señalarse durante la revisión y registrarse formalmente. Se revisan aspectos como consistencia, integridad, verificabilidad, comprensibilidad, rastreabilidad y adaptabilidad[28].
- ➢ **Prototipo de interfaz de usuario:** El prototipo de interfaz de usuario es una técnica de representación aproximada de la interfaz de usuario de un sistema software que permite a clientes y usuarios entender más fácilmente la propuesta de los ingenieros de requisitos para resolver sus problemas de negocio[26].

2.4 Diagrama de Clases:

Un diagrama de Clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas[29].

A continuación se muestra el diagrama de clases con estereotipos web identificado para el desarrollo del componente Gestión de servicios.

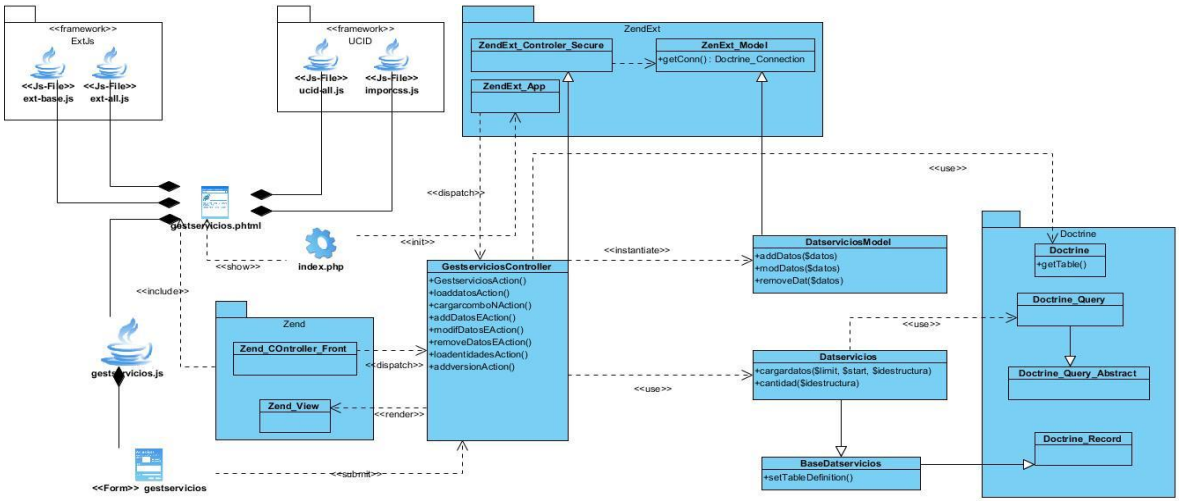


Figura 6: Diagrama de clases con estereotipos web.

Tabla 1: Descripción de las clases del diseño asociadas a la solución

Clases	Descripción
ZendExt_Controller_Secure	Clase de donde heredan todas las clases controladoras.
GestlserviciosController	Clase controladora que hereda de ZendExt_Controller_Secure. Encargada de realizar todas las funcionalidades del componente.
ZendExt_Model	Entre las características de esta clase se pueden mencionar que obtiene las conexiones activas en el sistema cuando se trata de acceder a una clase determinada del modelo
ZendExt_Exception	Clase que se encarga de manejar todas las excepciones.
Gestservicios.js	Formulario js a través del cual se maneja la información que introducen los usuarios del sistema.
Gestservicios.phtml	Plantilla HTML encargada de mostrarle al usuario la interfaz correspondiente al componente con la que va a trabajar.
<<Framework>> ExtJs	Librería JavaScript que utiliza la clase Gestlicense.phtml
DoctrineQuery	Permite la creación de consultas hacia la base de datos.
Doctrine_Query_Abstract	Es la clase usada para realizar consultas a la base de datos

	y presenta funcionalidades para filtrar los resultados de la consulta realizada y para determinar si un usuario tiene derecho a realizar la eliminación de un objeto de la base de datos, entre otras.
Index.php	Se encarga de determinar cuál es la clase controladora que le corresponde a cada página cliente.

2.5 Estilos arquitectónicos.

El estilo arquitectónico definido por el Departamento de Tecnología[30], donde se desarrolla el marco de trabajo SAUXE es el **Estilo N capas**. Donde se identificaron 3 niveles o capas fundamentales, como se muestra y describe a continuación.

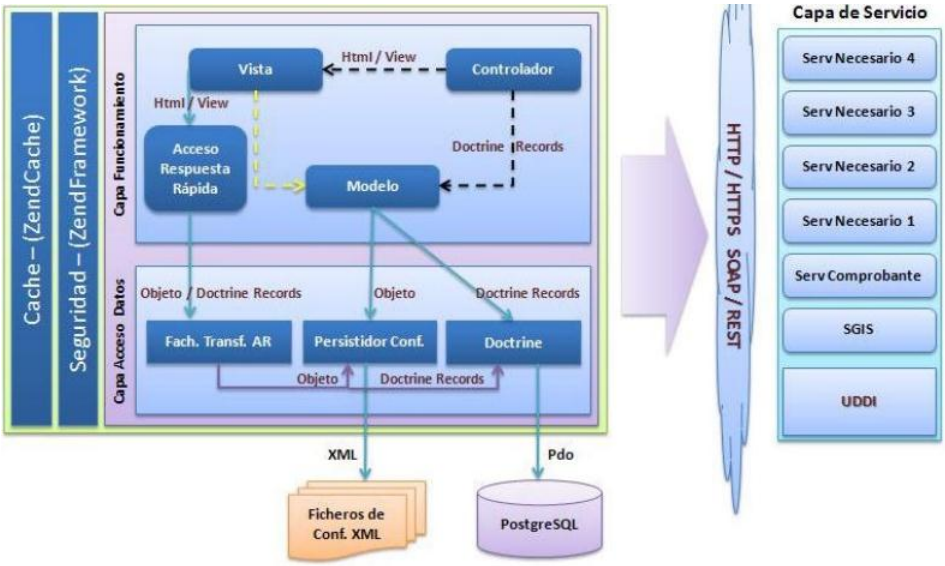


Figura 7: Estilo N Capas

- **Capa Datos:**

Donde estarán ubicados los servidores de base de datos PostgreSQL y MongoDB y además, un conjunto de Ficheros de Configuración del sistema.

- **Capa de Acceso a Datos:**

Donde estará presente el framework Doctrine, para la comunicación con el servidor de datos mediante el protocolo PDO[ix], también estará un marco de persistencia de datos que es el encargado de comunicarse vía XML [x]con los ficheros de configuración del sistema, y el último elemento de esta capa es la fachada de transferencia de acceso rápido cuya función es el acceso rápido tanto a los ficheros de comunicación como al servidor de datos, a través del marco de persistencia de datos de Configuración Doctrine para cada caso en específico.

- **Capa de Funcionamiento:** en esta capa se emplea el patrón de arquitectura Modelo Vista Controlador (MVC), así como el elemento de Acceso de Respuesta Rápida el cual brinda informaciones de forma vertiginosa violando las restricciones que impone el estilo en capas propuesto. De forma vertical al modelo descrito hasta este momento, estará el módulo de cacheo del sistema así como el encargado del tratamiento de la seguridad a nivel de aplicación Web.

2.5.1 Patrón Arquitectónico utilizado.

Como se explicó en el epígrafe anterior, el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** se emplea en la capa de funcionamiento y es el encargado de separar los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:[31]

- ✓ **Modelo:** Está compuesto por: datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el marco de persistencia de datos Doctrine.
- ✓ **Controlador:** Gestiona las entradas del usuario.
- ✓ **Vista:** Muestra la información del modelo usuario.

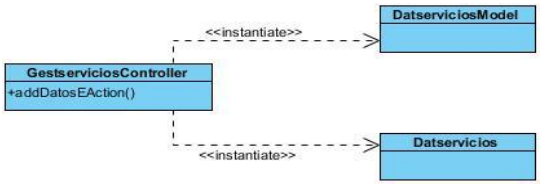
2.5.2 Patrones de diseño utilizados

Durante el desarrollo de la solución se utilizarán varios de los patrones generales de software para asignar responsabilidades (GRASP) así como patrones (GOF).

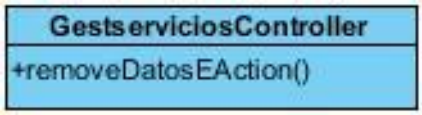
- **Patrones GRASP**

- ✓ **Experto:** Se puede ver el uso de este patrón en la clase *GestserviciosController* puesto que cada clase conoce su información y es la encargada de implementar las funcionalidades que brindan información de las mismas.

- ✓ **Creador:** Su uso se puede ver en la clase controladora GestserviciosController, ya que ella es la encargada de la creación de objetos de varias clases como DatserviciosModel y Datservicios.



- ✓ **Controlador:** La clase controladora *GestlserviciosController* se encarga de llevar el control de todos los eventos relacionados con el negocio. Implementa las funcionalidades que dan respuesta a las peticiones del usuario un ejemplo es eliminar datos.



- ✓ **Alta Cohesión:** El uso de éste patrón indica que la información almacenada en las clases debe ser coherente y relacionada a lo que se maneja en dichas clases y para esta solución se ve su uso igualmente en la clase controladora *GestserviciosController* puesto que es una clase que presenta responsabilidades y colabora con otras clases para dar solución a las tareas que surgen en el flujo del negocio.

- ✓ **Bajo Acoplamiento:** El uso de éste patrón se evidencia en la poca relación existente entre las clases que conforman el componente.

- **Patrones GoF**

✓ **El patrón creacional *Singleton* o Solitario**. Se puede ver su aplicación en el uso del IoC que es donde se encuentran los servicios que se van a consumir dentro entre los componentes que integran al marco de trabajo SAUXE, donde se garantiza que cada clase tenga una instancia única y permite que la misma sea accedida desde cualquier parte del sistema, en la solución no se va a implementar el *Singlenton* pero se consumen servicios del IoC como se puede apreciar en el siguiente ejemplo.

El uso de estos patrones garantizó asignar a cada clase la responsabilidad que le corresponde, obtener el menor número de relaciones y dependencias entre clases y aumentar las posibilidades de reusabilidad de las mismas. Todos estos elementos ayudaron a la confección de un diseño de la solución de gran claridad y entendimiento.

2.6 Modelo de Datos:

Un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, permite describir el tipo de datos que incluye la base y la forma en que se relacionan, las restricciones de integridad y las operaciones de manipulación de los datos[32].

Para la herramienta de control de licencias de software se genera el siguiente modelo de datos

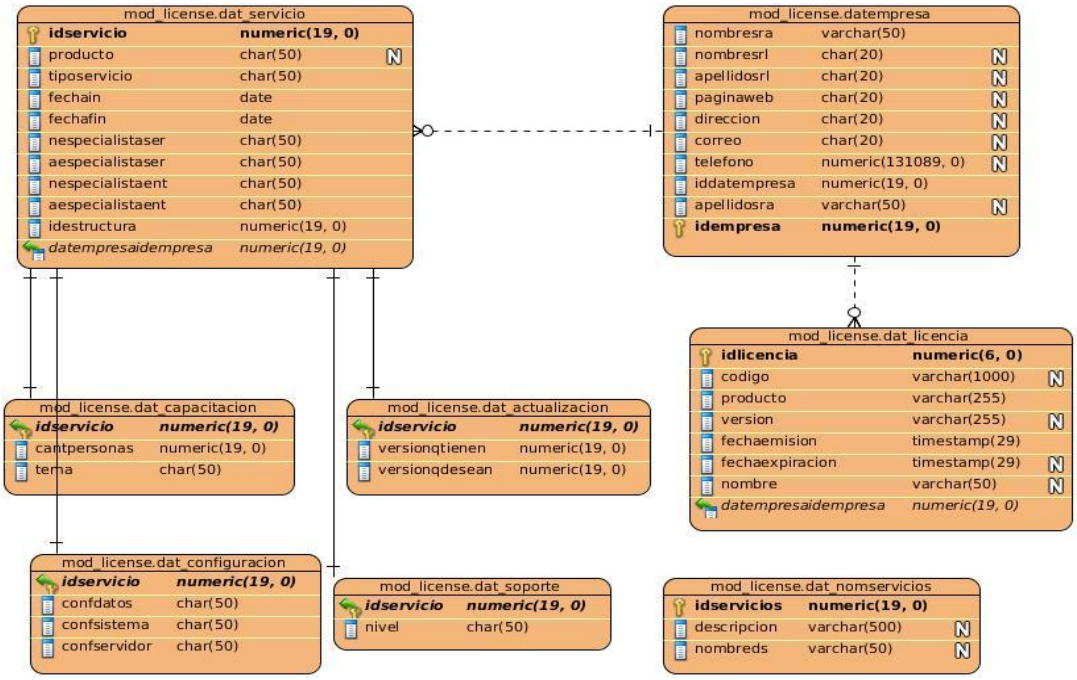


Figura 8 Modelo de datos

En el modelo de datos anterior se encuentran las siguientes tablas siendo las siguientes las de más importancia:

- ✓ ***dat_nomservicios*:** Es una clase nomencladora de servicios.
- ✓ ***dat_licencia*:** Es la clase encargada de la creación de las licencias.
- ✓ ***datempresa***: Es la clase encargada de recoger todos los datos asociados a una entidad.
- ✓ ***dat_servicio*:** Es la clase encargada de la gestión de servicios.

2.7 Conclusiones del Capítulo:

En éste capítulo se elaboró el modelo conceptual de la solución, donde se identificó la relación que debía existir entre cada uno de los conceptos identificados. Se realizó la descripción de los requisitos funcionales y no funcionales identificados durante el proceso de Ingeniería de requisitos, validando los mismos con técnicas de validación de requisitos asociadas a prototipos de interfaz y revisión técnica. Como parte del modelo de diseño elaborado, se estudió el estilo arquitectónico definido para el desarrollo en el departamento y se identificaron los patrones de diseño a utilizar en el desarrollo de la solución. Por

último, se elaboró y describió el diagrama clases y el modelo de datos con que contará la solución. Logrando de esta manera sentar las bases para la implementación de la propuesta de solución.

Capitulo III: Implementación y validación.

3.1 Introducción:

En el presente capítulo se muestran algunas métricas que se aplican actualmente para validar la calidad en el diseño de software y se definen cuáles se aplicaron al diseño de la solución propuesta. Estas proporcionan una medida de cuán evolucionado se encuentra el desarrollo de la aplicación informática desde la visión interna que proporcionan los parámetros que estas definen. Además se describen las pruebas de aceptación realizadas al software y sus resultados.

3.2 Diagrama de componentes.

Los diagramas de componentes describen los elementos físicos (o componentes) del sistema y sus relaciones, mientras que los componentes representan todo tipo de elementos que serán desarrollados; pueden ser simples archivos, paquetes, entre otros. Para el desarrollo del componente para la gestión de licencias de software para las soluciones del Centro de Informatización de la Gestión de Entidades es necesaria la interacción con otros componentes del marco de trabajo como son: ZendExt, Doctrine y ExtJS. El mismo hace uso de servicios internos mediante el IoC para obtener información de otros subsistemas del marco de trabajo, entre ellos los que brinda Estructura y Composición. A continuación se muestra el diagrama de componentes elaborado.

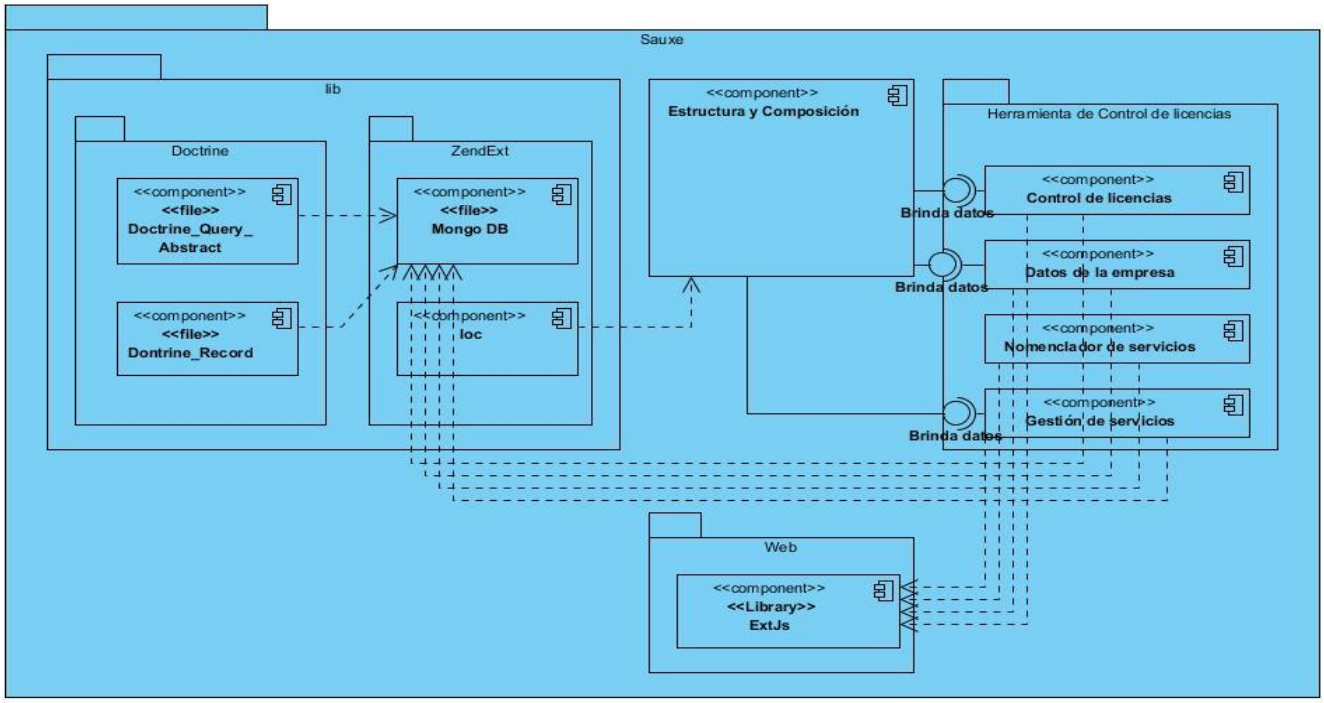


Figura 9: Diagrama de componentes

3.3 Diagrama de Despliegue.

El Diagrama de Despliegue se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes (nodos). Los nodos son objetos físicos que existen en tiempo de ejecución y que representan algún tipo de recurso computacional, también pueden ser dispositivos del sistema. El usuario accede, desde su puesto de trabajo al sistema que se encuentra instalado en el servidor de aplicaciones y dicho servidor se conecta a los servidores de base de datos, para realizar las consultas y obtener los resultados deseados. A continuación se muestra el diagrama de despliegue elaborado.

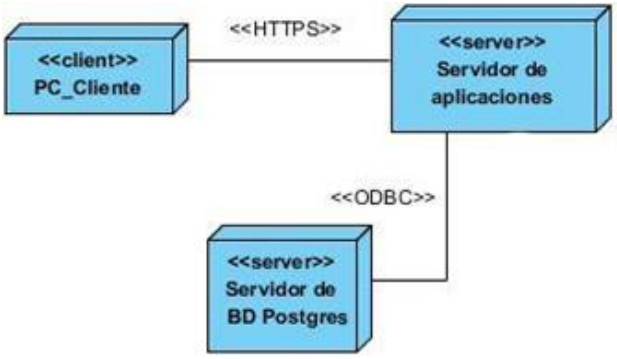


Figura 10 Diagrama de despliegue

3.4 Estándar de codificación.

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. El uso de estándares de codificación permite lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo[27].

Para el desarrollo del componente administración de conexiones se utilizarán algunos de los estándares de codificación y normas propuestos como parte de la Línea de arquitectura determinada para el desarrollo del ERP Cuba[27].

Los estándares utilizados en la codificación fueron los siguientes:

- **Notación Húngara:** Escomentariarta convención se basa en definir prefijos para cada tipo de datos y según el ámbito de las variables. También es conocida como notación REDDICK (por el nombre de su creador). La idea de esta notación es la de dar mayor información al nombre de la variable, método o función definido en ella un prefijo que indique su tipo de dato o ámbito.[33]
- **Notación PascalCasing:** Es como la notación húngara pero sin prefijos. En este caso los identificacdores y nombres de las variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula[34].
- **Notación CamelCasing:** Es parecido al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula[35].
- **Nomenclatura de las clases.**

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing. Con sólo leerlo se reconoce el propósito de la misma.

Ejemplo: *DatServicios.*

- **Nomenclatura según el tipo de clases.**

1. **Clase controladora.**

Las clases controladoras después del nombre llevan la palabra: “Controller”.

Ejemplo: *GestlicenseController*.

2. **Clases de los modelos.**

- ✓ **Business (Negocio).**

Las clases que se encuentran dentro de Business después del nombre llevan la palabra: “Model”.

Ejemplo: *DatserviciosModel*

- ✓ ***Domain* (Dominio).**

Las clases que se encuentran dentro de *Domain* el nombre que reciben es el de la tabla en la Base de Datos.

Ejemplo: *Datservicios*

- ✓ ***Generated* (Dominio bases)**

Las clases que se encuentran dentro de *Generated* el nombre comienza con la palabra: “Base” y seguido el nombre de la tabla en la Base de Datos.

Ejemplo: *BaseDatservicios.*

- **Nomenclatura de las funciones.**

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido del sufijo “Action”.

Ejemplo: *removelicenseAction.*

- **Nomenclatura de las variables.**

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y comenzando con un prefijo según el tipo de datos.

Ejemplo: *$license.*

- **Prefijos para los tipos de datos.**

Los prefijos a utilizar en la creación de variables serán los siguientes:

Tabla 2: Prefijos para los tipos de datos

Tipos de Datos	*Prefijos*
Arreglos	arr
Objetos	obj
Enteros	int
Cadena	str
float	flt
Boolean	Bool

- **Normas de comentariado.**

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se pueda aumentar su mantenimiento a lo largo del tiempo.

- **Nomenclatura de los comentarios.**

Los comentarios deben ser lo bastante claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando.

1. **En las clases**

Antes de la declaración de una clase se escribe una breve descripción donde se explique el propósito de la misma. Se escribe de la siguiente forma:

```
/**
* Nombre de la clase *
* Descripcion *
* @author *
* @package *(módulo)
* @subpackage *(sub módulo)
* @copyright *
* @version (versión - parche) */
```

2. **En las funciones**

Antes de la declaración de la función se escribe una breve descripción donde se explique el propósito de la misma. Se escribe de la siguiente forma:

```
/**
* Nombre de la función *
* Descripcion *
```

* @author * (en caso de que no sea el autor de la clase)
* @param *(los parámetros que se le pasan a la función con su descripción)
* @throws *(en caso de que dispare una excepción)
* @return *(se pone lo que devuelve la función y un comentario) */

- **Estilo del código.**

En la implementación cuando se escriba una sentencia en php la forma de utilizar los tab del mismo es la siguiente:

```
<?php
//código
?>
```

- **Sangría o Indexado**

La política de sangría a utilizar en la implementación es por **tab**.

Ejemplo:

```
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
    }
    }
    function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
    }
}
?>
```

- **Brazas o llaves.**

En la declaración de clases o interfaces, métodos y bloques, la apertura de llaves se hace en la misma línea.

Ejemplo:

```php
<?php
/**
 * Braces
 */
interface EmptyInterface {
}

class Example {
    function bar($p) {
        for($i = 0; $i < 10; $i ++) {
        }
        switch ( $p) {
            case 0 :
                $fField->set ( 0 );
            break;
            case 1 :
                {
                    break;
                }
            default :
                $fField->reset ();
        }
    }
}
?>
```

- **Espacios en blanco.**

1. **Arreglos.**

La declaración de los espacios en blanco en los arreglos es como se muestra en el ejemplo.

Ejemplo:

```php
<?php
class MyClass implements I0, I1, I2 {
}
class MyClass {
    public $a = 0, $b = 1, $c = 2, $d = 3;
    const MY_TRUE = 1, MY_FALSE = 2;
}
function foo() {
}
function bar(int $x, $y, $z = 1) {
}
?>
```

3.5 Métricas de Diseño.

La evaluación de un producto, mediante métricas, es un aspecto fundamental a tener en cuenta; ya que, aunque las métricas del producto el software no suelen ser absolutas, brindan la posibilidad de evaluar la calidad a partir de varias reglas definidas claramente. Permiten detectar y corregir los posibles problemas que se puedan presentar durante el proceso de desarrollo y no después de terminado el producto[36].

La calidad del diseño se puede evaluar aplicando métricas básicas para la calidad del diseño orientado a objetos. Los atributos de calidad involucrados son:

- **Responsabilidad**: responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** grado de complejidad que posee la implementación de un diseño de clases específico.
- **Reutilización**: grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento**: grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- **Complejidad del mantenimiento:** grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas**: número o grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, modulo, clase, conjunto de clases, entre otros) diseñado.

Las métricas seleccionadas para evaluar la calidad del diseño de la Herramienta para la gestión de licencias de software para las soluciones del Centro de Informatización de la Gestión de Entidades (CEIGE) son:

3.5.1 Métrica Tamaño Operacional de Clases (TOC):

Está dado por el número de métodos u operaciones (de instancia privada y heredada) que están encapsulados dentro o por una clase[37]. Evalúa los siguientes atributos de calidad:

Tabla 3: Atributos de calidad que evalúa TOC.

Atributo de Calidad	Modo en que lo afecta
Responsabilidad	Aumento del TOC provoca aumento de la responsabilidad asignada a la clase.
Complejidad de Implementación	Aumento del TOC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del TOC provoca disminución del grado de reutilización de la clase.

Para la evaluación de dichos atributos de calidad, se definieron los siguientes criterios y categorías de evaluación:

Tabla 4: Criterios de evaluación de la métrica TOC.

Atributo	Categoría	Criterio
Responsabilidad	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad de Implementación	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio

Tabla 5: Instrumento de evaluación de la métrica TOC.

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
GestlicenseController	8	Media	Media	Media
GestdatosempresaController	6	Media	Media	Media
GestnomserviciosController	5	Baja	Baja	Alta
GestserviciosController	8	Media	Media	Media
DatLicenciaModel	9	Media	Media	Media
DatempresaModel	3	Baja	Baja	Alta
DatnomserviciosModel	3	Baja	Baja	Alta
DatserviciosModel	3	Baja	Baja	Alta

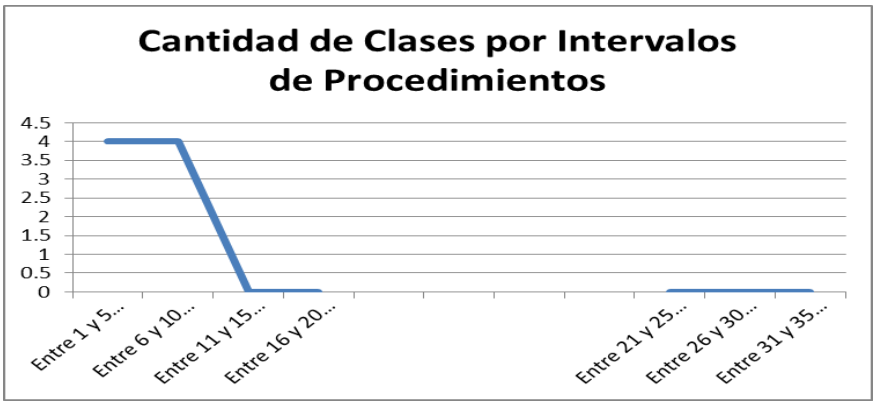


Figura 11: Representación de las clases según la cantidad de operaciones.

Teniendo en cuenta el umbral de cantidad de operaciones con que cuenta cada clase, está establecido que un tamaño de clase pequeño es aquel que tiene un valor menor o igual que 3, un tamaño medio para aquellas clases que tengan entre 4 y 12 operaciones y un tamaño de clase grande es aquel que es mayor o igual que 16. Se concluye que el componente cuenta con 4 clases fundamentales cuyo promedio de cantidad de operaciones equivale a 6. Los valores de tamaño quedan distribuidos de la siguiente manera:

Tabla 6: Tamaño de clases

Umbral	Tamaño	Cantidad de Clases
Pequeño	<=3	3
Medio	>=4 y <=12	5
Grande	>=16	0

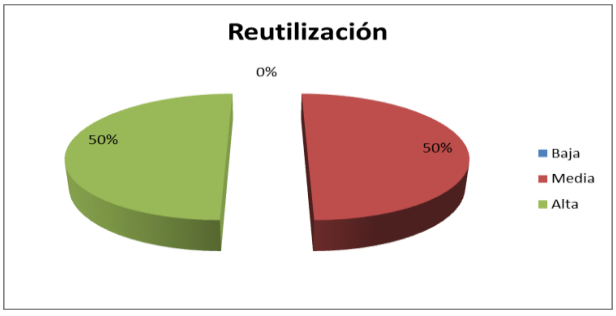


Figura 12: Resultados de la evaluación de la métrica TOC para el atributo de calidad Reutilización.

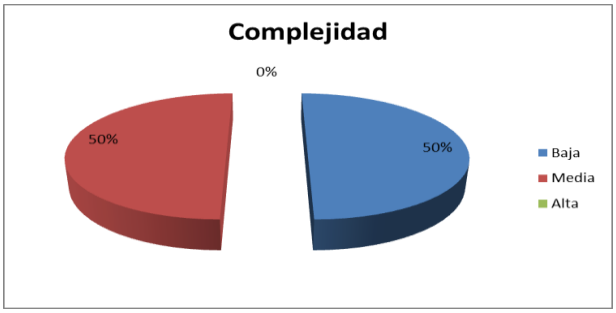


Figura 13: Resultados de la evaluación de la métrica TOC para el atributo de calidad Complejidad.

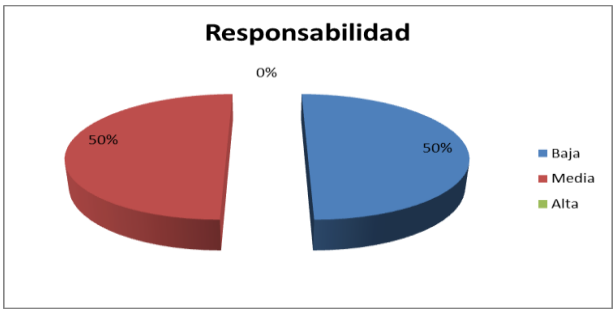


Figura 14: Resultados de la evaluación de la métrica TOC para el atributo de calidad Responsabilidad.

Resultados Obtenidos.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el de la herramienta para el control del versionado de los productos del CEIGE tiene una calidad aceptable teniendo en cuenta que el 50% de las clases poseen evaluaciones positivas en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).

3.5.2 Métrica Relación entre Clases (RC):

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad[38]:

Tabla 7: Atributos de calidad que evalúa RC

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Aumento del RC provoca aumento del Acoplamiento de la clase.

Complejidad de mantenimiento	Aumento del RC provoca aumento de la complejidad del mantenimiento de la clase.
Reutilización	Aumento del RC provoca disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Aumento del RC provoca aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Para la evaluación de dichos atributos de calidad, se definieron los siguientes criterios y categorías de evaluación:

Tabla 8: Criterios de evaluación de la métrica RC.

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Baja	1
	Media	2
	Alta	>2
Complejidad de mantenimiento	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

Tabla 9: Instrumento de evaluación de la métrica RC.

Clases	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant	Reutilización	Cantidad de Pruebas
GestlicenseController	4	Alta	Media	Media	Alta
GestdatosempresaController	4	Alta	Media	Media	Alta
GestnomserviciosController	4	Alta	Media	Media	Alta
GestserviciosController	4	Alta	Media	Media	Alta
DatLicenciaModel	1	Baja	Baja	Alta	Baja
DatempresaModel	1	Baja	Baja	Alta	Baja
DatnomserviciosModel	1	Baja	Baja	Alta	Baja
DatserviciosModel	1	Baja	Baja	Alta	Baja

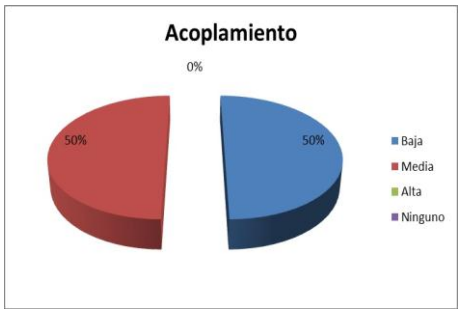


Figura 15: Resultados de la evaluación de la métrica RC para el atributo de calidad Acoplamiento.

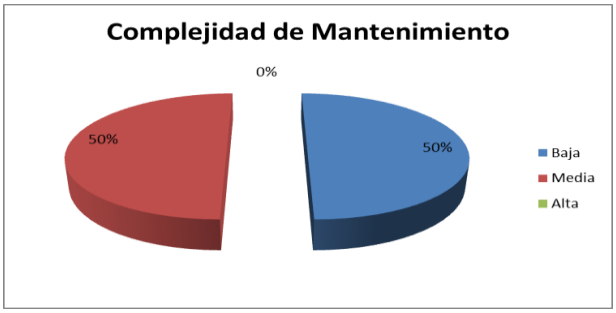


Figura 16: Resultados de la evaluación de la métrica RC para el atributo de calidad Complejidad de Mantenimiento.

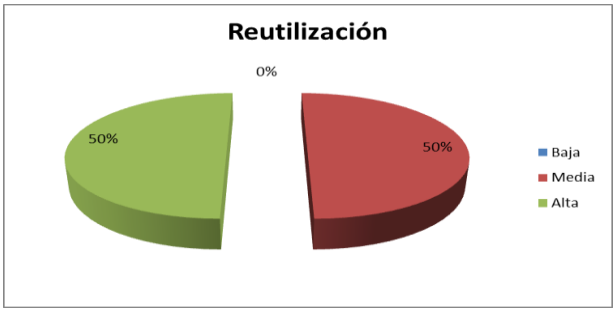


Figura 17: Resultados de la evaluación de la métrica RC para el atributo de calidad Reutilización.

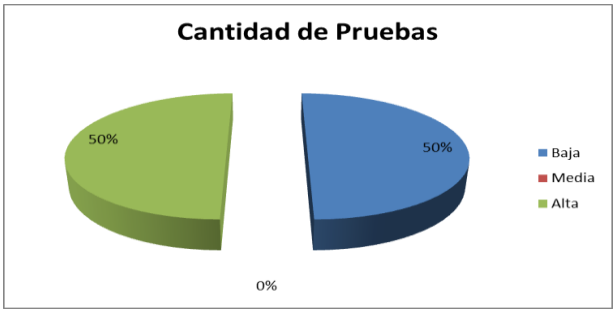


Figura 18: Resultados de la evaluación de la métrica RC para el atributo de calidad Cantidad de Pruebas.

Resultados obtenidos

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño de la herramienta para el control del versionado de los productos del CEIGE tiene una calidad aceptable teniendo en cuenta que el 50 % de las clases posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 50 % de las clases.

3.5.3 Matriz interferencia de indicadores de calidad:

En ésta investigación se define que el 50% las clases presentan Responsabilidad, Complejidad y Acoplamiento clasificados de Media; 50% de las clases exhiben Complejidad de mantenimiento y Cantidad de pruebas clasificados de Alta y el 50% de las clases muestran Reutilización Alta o Baja, serán considerados resultados Favorables cuando los atributos mencionados anteriormente cuenten con un porcentaje del 50% o superior y por debajo de 50% se consideraran resultados no favorables.

La matriz de interferencia de indicadores de calidad es una representación de los atributos de calidad y las métricas utilizadas para evaluar la calidad del diseño propuesto para el componente. Con ella se puede conocer si los resultados obtenidos de las relaciones atributo/métrica son positivo o no, llevando estos resultados a una escalabilidad numérica donde, si los resultados son positivos se le asigna el valor 1 si son negativos toma valor 0 y si no existe relación es considerada como nula y es representada con un guión simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas.

Tabla 10: Resultados de la evaluación de la relación Atributo/Métrica

Atributo/Métrica	TOC	REC	Promedio
Responsabilidad	1	-	1
Complejidad de Implementación	1	-	1
Reutilización	1	1	1
Acoplamiento	-	1	1
Complejidad de Mantenimiento	-	1	1
Cantidad de pruebas	-	1	1

Teniendo en cuenta los resultados obtenidos al aplicar las métricas de diseño TOC y RC, que se muestran en las **Tabla 3** y **Tabla 8**, se construye el siguiente gráfico (**Ver figura 19**.), donde se muestran los resultados obtenidos al evaluar los atributos de calidad involucrados en cada una de las métricas aplicadas.

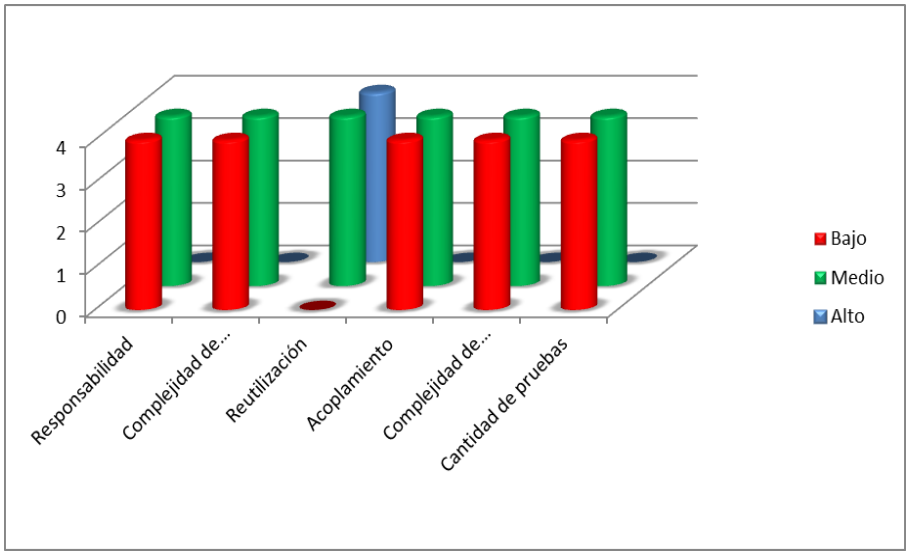


Figura 19: Resultados de la evaluación.

3.6 Pruebas de Software.

Las pruebas de software son un instrumento para determinar el status de la calidad de un software. Tienen como objetivo, además de descubrir errores, medir el grado en que el software cumple con los requerimientos definidos[39] .

3.6.1 Pruebas de Caja Blanca:

Las pruebas de caja blanca son un método de diseño que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba. Al emplear sus métodos, se puede garantizar que, al menos una vez, se ejecuten todas las rutas independientes del módulo.[40]

La prueba del Camino Básico es una técnica que permite derivar casos de prueba a partir de un conjunto de caminos independientes por los cuales puede circular el flujo de control[40]. Para obtener el conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su Complejidad ciclomática[41] .

Para poder elaborar el Grafo de Flujo, primero se deben enumerar las sentencias del código:

```
function loadentidadesAction(){
        $security = ZendExt_Aspect_Security_Sgis::getInstance();(1)
        $idestructuracomun = $this->_request->getPost('node');(1)
        $dominios = $security->getDomain($idestructuracomun);(1)
    foreach ($dominios as $key => $value) {(2)
        $estructura = $this->integrator->metadatos->DameEstructura($value['id']);(3)
        $obj  = new ZendExt_Nomencladores_ADT();(3)
        $datosDpa = $obj->getElement("nom_dpa",$estructura[0]->iddpa);(3)
        $datosEsp = $obj->getElement("nom_especialidad",$estructura[0]->idespecialidad);(3)
        $datosDpa1 = $obj->getElement("nom_dpa",$datosDpa[idpadre]); (3)
        $dominios[$key]['provincia'] = $datosDpa1['denominacion']; (3)
        $dominios[$key]['municipio'] = $datosDpa['denominacion'];(3)
        $dominios[$key]['especialidad'] = $datosEsp['denespecialidad']; (3)
    } (4)
    if (count($dominios))(5)
        echo json_encode($dominios);(6)
    else
        echo json_encode(array());(7)
}(8)
```

Figura 20: Sentencias de código

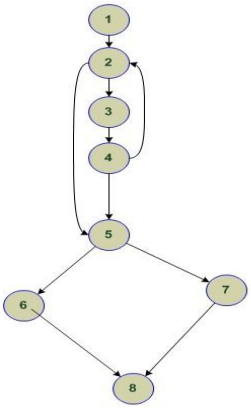


Figura 21: Grafo de flujo

La complejidad ciclomática es la métrica de software con que se define la cantidad de caminos independientes de cada una de las funcionalidades del programa y provee el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. [40]

Según Pressman la complejidad ciclomática se basa en la teoría gráfica y se calcula de 3 maneras distintas, donde, para que el cálculo sea correcto, todas deben arrojar el mismo resultado[40]:

1. El número de regiones corresponde a la complejidad ciclomática "V (G)".

V (G) = R **Donde R es la cantidad total de regiones**.

V (G) = 4

2. V (G) = A – N + 2: **Donde A es el número de aristas y N número de nodos.**

V (G) = 11 – 9 + 2

V (G) = 4

3. V (G) = P + 1: **Donde P es el número de nodos predicados.**

V (G) = 3 + 1

V (G) = 4

Teniendo en cuenta el valor obtenido de la complejidad de la funcionalidad cargar entidades, se obtuvo la cantidad de posibles caminos independientes y cantidad de pruebas que se deben realizar para comprobar que cada sentencia se ejecute al menos una vez.

El cálculo arrojó que V (G) = 3, por lo que los posibles caminos básicos son:

- ✓ Camino básico Nº 1: 1, 2, 5.
- ✓ Camino básico Nº 2: 1, 2, 3, 4, 2.

- ✓ Camino básico Nº 3: 1, 2, 3, 4, 5, 6, 8.
- ✓ Camino básico Nº 4: 1, 2, 3, 4, 5, 7, 8.

➢ **Derivación de casos de prueba para la funcionalidad *loadentidades*.**

Luego de elaborados el Grafo de Flujo y los caminos básicos a recorrer, se elaboran los casos de prueba correspondiente a cada camino con los que se garantizará que cada sentencia de código se ejecute al menos una vez en cada caso identificado.

- ✓ **Caso de prueba para camino básico Nº 1-** (1, 2, 5).
  Variable de entrada = $dominio.
  Si $dominio = vacío y count($dominio).
- ✓ **Caso de prueba para camino básico Nº 2-** (1, 2, 3, 4, 2).
  Variable de entrada = $dominio.
  Mientras $dominio != vacío.
- ✓ **Caso de prueba para camino básico Nº 3-** (1, 2, 3, 4, 5, 6, 8).
  Variable de entrada = $dominio.
  Mientras $dominio != vacío.
- ✓ **Caso de prueba para camino básico Nº 4-** (1, 2, 3, 4, 5, 7, 8).
  Variable de entrada = $dominio.
  Mientras $dominio != vacío.

3.6.2 Pruebas de Caja Negra:

Las pruebas de caja negra son las que se aplican a la interfaz del software y se centran en los requisitos funcionales del sistema; permitiendo derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa.[33]

Para realizarle las pruebas de caja negra la herramienta para el control del versionado de los productos del CEIGE, se elaboraron casos de pruebas específicos para cada uno de los RF descritos, los mismos se encuentran en el expediente de proyecto de Sauxe$_{v2.2}$ en la carpeta Diseños Casos de Prueba y en el caso de los casos de prueba de la Gestión de licencias se muestran en el (**Anexo 1**). Dichos casos de prueba fueron evaluados y aprobados por el equipo de trabajo del Dpto. de Tecnología encargado de revisar y liberar el componente para ser usado como parte de Sauxe, para ello se efectuaron 2 iteraciones de

pruebas, obteniendo en la primera 8 no conformidades de nomenclatura las mismas se resuelven y en una segunda iteración se encuentran 0 (cero) no conformidades.

3.7 Validaciones técnicas de la solución.

La herramienta para la gestión del control de las soluciones del CEIGE fue revisada por especialistas del departamento de tecnología logrando cumplir con los requisitos planificados para la implementación de la misma, de esta manera, se entrega un acta de liberación del departamento de tecnología y otra proveniente del cliente de la solución. Además, para el despliegue del proyecto Cedro ERP se generó una llave con la cual se instaló el sistema y mantiene un funcionamiento satisfactorio. En el (Anexo 2) se muestra el acta de liberación del departamento de tecnología y del cliente como constancia de su funcionamiento.

3.8 Conclusiones del Capítulo.

En éste capítulo se realizó el modelo de implementación y como parte de él se obtuvieron el Diagrama de Componentes y de Despliegue de la solución y se documentaron los estándares de codificación por los que se guió la implementación del componente.

Asimismo se realiza un análisis de las métricas necesarias para comprobar la calidad del software obteniendo resultados satisfactorios de los atributos de calidad asociados a reutilización, acoplamiento, complejidad de implementación y mantenimiento así como cantidad de pruebas.

También se realizaron pruebas funcionales, en el caso de caja negra especialistas del departamento de tecnología del CEIGE efectuaron 2 iteraciones de pruebas, obteniendo en la última 0 (cero) no conformidades.

CONCLUSIONES GENERALES

Al concluir la investigación para el desarrollo de la herramienta para la gestión de licencias de software para las soluciones del Centro de Informatización de la Gestión de Entidades (CEIGE) se pudo arribar a las siguientes conclusiones:

- Se consolidó el basamento teórico con vistas al desarrollo del software mediante un estudio del estado del arte de las principales herramientas que son usadas para la gestión del licenciamiento de software obteniendo como resultado del estudio, que ninguna de las herramientas estudiadas cumple con las necesidades del CEIGE por lo que se decide desarrollar una herramienta que gestione el control de los activos de software del centro. Se define el modelo de desarrollo a seguir, el cual describe cada artefacto a obtener en cada etapa del desarrollo de esta aplicación y así se sientan las bases para su inicio.
- Se elaboró el modelo conceptual de la solución a partir de entrevistas con especialistas del departamento de tecnología, ALBET y el Centro de soporte. Se realizó la descripción de los requisitos funcionales y no funcionales validando los mismos con técnicas de validación de requisitos. Como parte del modelo de diseño elaborado, se estudió el estilo arquitectónico definido para el desarrollo en el departamento y se identificaron los patrones de diseño a utilizar en el desarrollo de la solución. Por último, se elaboró y describió el diagrama clases y el modelo de datos con que contará la solución.
- Se realizó el modelo de implementación obteniendo el Diagrama de Componentes y de Despliegue de la solución. Se documentaron los estándares de codificación por los que se guía la implementación del componente. Asimismo se realiza un análisis de las métricas necesarias para comprobar la calidad del software obteniendo resultados satisfactorios de los atributos de calidad asociados a reutilización, acoplamiento, complejidad de implementación y mantenimiento así como cantidad de pruebas. También se realizaron pruebas funcionales, en el caso de caja negra especialistas del Dpto. de tecnología del CEIGE efectuaron 2 iteraciones de pruebas, obteniendo en la última 0 (cero) no conformidades.
- Por último, para probar que el componente constituye una mejora el control de los activos de software del CEIGE, se generó una llave para el proyecto Cedro ERP con la cual se instaló el sistema y mantiene actualmente un funcionamiento normal logrando así los resultados esperados.

Además se obtiene una carta de liberación otorgada por especialistas del departamento y otra proveniente del cliente asociado a la solución.

RECOMENDACIONES

- Incluir las funcionalidades que permitan el manejo de los tipos de licencias.
- Asociar los servicios a determinados tipos de licencias.

BIBLIOGRAFÍA

[1] «Microsoft Home Page | Devices and Services». [En línea]. Disponible: http://www.microsoft.com/en-us/default.aspx. [Accedido: 26-ene-2013].
[2] D. Martín Toral, *Prácticas del Módulo Aplicaciones Ofimáticas*. 2011.
[3] «» Protección de código: llaves HASP SRM >> MKM Publicaciones». [En línea]. Disponible: http://www.mkm-pi.com/isv-magazine/proteccion-de-codigo-llaves-hasp-srm/. [Accedido: 27-ene-2013].
[4] «Productos NetSupport». [En línea]. Disponible: http://www.softart.cl/index.php/netsupport%20-dna-inventario. [Accedido: 27-ene-2013].
[5] «ElecKey 2.0 The Complete Solution for Software Protection, Software Copy Protection, Software Licensing, and Secure Software Distribution». [En línea]. Disponible: http://www.sciensoft.com/products/eleckey/. [Accedido: 27-ene-2013].
[6] «Mirage Computer Systems GmbH: Licence Protector». [En línea]. Disponible: http://www.licence-protector.com/. [Accessed: 27-ene-2013].
[7] Danairy Averoff Cabrera María Luisa Enriquez Pérez, «Diseño de una herramienta para la gestión del otorgamiento de licencias para el sistema alas PACSViewer», 2010.
[8] Ing. William Gonz ález Obreg ón, «CEIGE-Modelo de desarrollo de software v1.0», 2012.
[9] «UML, BPMN and Enterprise Architecture Tool for Software Development». [En línea]. Disponible: http://www.visual-paradigm.com/. [Accedido: 25-ene-2013].
[10] «Control de versiones con Subversion». [En línea]. Disponible: http://svnbook.red-bean.com/index.es.html. [Accedido: 26-ene-2013].
[11] «RapidSVN». [En línea]. Disponible: http://www.rapidsvn.org/. [Accedido: 26-ene-2013].
[12] «EMS SQL Manager - PostgreSQL Tools - Products Family». .
[13] «Welcome to NetBeans». [En línea]. Disponible: http://netbeans.org/. [Accessed: 25-ene-2013].
[14] Baryolo, O.G, «SOLUCIÓN INFORMÁTICA DE AUTORIZACIÓN EN ENTORNOS MULTIENTIDAD Y MULTISISTEMA», 2001.
[15] «ExtJS lo bueno, lo malo y lo feo | Desarrollo en Web». .
[16] Pedro Boda, K.N., Javier Martinez, Benjamín Gonzales, *Zend Framework Manual en Español*. 2009.
[17] Smith, L, *Introduction to the Doctrine Object Relational Mapper*. 2009.
[18] Larman, C, *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 2011.
[19] «PHP: Hypertext Preprocessor». [En línea]. Disponible: http://php.net/. [Accedido: 27-ene-2013].
[20] «Acerca de JavaScript - JavaScript | MDN». [En línea]. Disponible: https://developer.mozilla.org/es/docs/JavaScript/Acerca_de_JavaScript. [Accessed: 27-ene-2013].
[21] Christopher Alexander, «Patrones», .
[22] «Patrones de diseño. Análisis y Diseño. Ingenieria del Software.» .
[23] Martínez, J.S, *Una Arquitectura para una Herramienta de Diseño*. Universidad de Murcia: Departamento de Informática y Sistema.: .
[24] «El Patrón Singleton». .
[25] Pérez, Ing. Mileidy M. Sarduy, «Modelo Conceptual (CIG-SEG-N-i1301). Departamento de Tecnología, Universidad de las Ciencias Informáticas». 2012.
[26] IEEE, *Introducción a la Ingeniería de Requisitos. Ingeniería de software.* 2010.
[27] Damian Alfonso, «Los estándares de codificación». .
[28] ADDISON-WESLEY, *Ian Sommerville, Software Engineering,*. .
[29] «Diagramas de Clases y Objetos». .
[30] Mena López, Grettel Leydi y Tenrrero Cabrera, Marianela., «Arquitectura ERP.» .
[31] «How to use Model-View-Controller (MVC)». .

[32] Massachusetts Institute of Technology, «Métodos de desarrollo de software». 2002.
[33] «INFORMACIÓN: Objeto notación húngara convenciones de nomenclatura para VB». .
[34] «http://www.informatizate.net - notacion PascalCasing». [En línea]. Disponible: http://www.informatizate.net/articulos/dime_como_programas_y_te_dire_quien_eres_23082004.html. [Accedido: 22-may-2013].
[35] «Notación y estilo en programación». .
[36] «Métricas de Software». .
[37] «Métrica de diseño - EcuRed». .
[38] «Métricas de Software». .
[39] «PRUEBASDESOFTWARE Inicio: Cursos y servicios de gestión de calidad de software». .
[40] Pressman, Roger S, *Software Engineering. A practitioner´s Approach. 7th Edition.* .
[41] «Caja blanca, caja negra». .

ANEXOS

1.1 Anexo 1

Adicionar Licencia

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1: Adicionar licencia.	Se adiciona una licencia.	EP 1.1: Adicionar licencia.	– Se selecciona una entidad. – Se presiona el botón **Adicionar.** – **Se abre la interfaz Adicionar.** – Se insertan los datos del usuario. – Se presiona el botón **Aceptar**. – Se muestra un mensaje indicando el éxito de la operación.

		EP 1.2: Adicionar licencia dejando campos requeridos en blanco.	– Se selecciona una entidad. – Se presiona el botón **Adicionar.** – **Se abre la interfaz Adicionar.** – Se introducen los datos dejando campos
		EP 1.3: Adiciona licencia introduciendo error en los datos.	– Se selecciona una entidad. – Se presiona el botón **Adicionar.** – **Se abre la interfaz Adicionar.** – Se introducen los datos del usuario
		EP 1.4: Aplicar	– Se selecciona una entidad. – Se presiona el botón **Adicionar.** – **Se abre la interfaz Adicionar.**

		EP 1.5: Cancelar	– Se presiona el botón **Adicionar.** – **Se abre la interfaz Adicionar usuario.** – Se introducen o no los datos en el formulario. – Se presiona el botón **Cancelar**

Modificar licencia

Nombre del requisito	**Descripción general**	**Escenarios de pruebas**	**Flujo del escenario**
1: Modificar licencia.	Se modifica una licencia.	EP 1.1: Modificar licencia.	– Se selecciona la licencia que desea modificar. – Se presiona el botón **Modificar.** – **Se abre la interfaz Modificar usuario.** – Se modifican los datos deseados. – Se presiona el botón **Aceptar**. – Se muestra un mensaje indicando el éxito de la operación

		EP 1.2: Modificar licencia dejando campos requeridos en blanco.	– Se selecciona la licencia que desea modificar. – Se presiona el botón **Modificar.** – **Se abre la interfaz Modificar usuario.** – Se introducen o modifican los datos dejando campos requeridos en blanco. – Se presiona el botón **Aceptar**. – Se muestra un mensaje de error indicando que hay campos vacíos.
		EP 1.3: Modificar licencia introduciendo error en los datos.	– Se selecciona el servicio que se desea modificar. – Se presiona el botón **Modificar.** – **Se abre la interfaz Modificar usuario.** – Se introducen o modifican los datos de un usuario insertando errores en los datos. – Se presiona el botón **Aceptar**. – Se muestra un mensaje indicando la existencia de

		EP 1.4: Cancelar	– Se selecciona el servicio que se desea modificar. – Se presiona el botón **Modificar.** – **Se abre la interfaz Modificar usuario.** – Se modifican o no los datos de un usuario. – Se presiona el botón **Cancelar** para abortar la operación. – El sistema cancela la acción.

Eliminar licencia

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1. Eliminar licencia.	Se elimina la licencia seleccionada.	EP1.1: Eliminar licencia.	- Se selecciona una licencia. - Se presiona el botón **Eliminar**. - El sistema muestra un mensaje de confirmación: "¿Está seguro que desea eliminarla(s)?" - Se presiona el botón **Aceptar**. - Se muestra un mensaje indicando el éxito de la operación.

		EP 1.2: Cancelar.	- Se selecciona la licencia. - Se presiona el botón **Eliminar**. - El sistema muestra un mensaje de confirmación: "¿Está seguro que desea eliminarla(s)?" - Se presiona el botón **Cancelar** para abortar la operación. - El sistema cancela la acción.

Descargar Licencia

Nombre del requisito	Descripción general	Escenarios de pruebas	Flujo del escenario
1. Descargar licencia.	Se descarga la licencia seleccionada.	EP1.1: Descargar licencia.	- Se selecciona una licencia. - Se presiona el botón **Descargar**. - El sistema muestra una interfaz para que el usuario escoja el destino - Se presiona el botón **Aceptar**. - Se muestra un mensaje indicando el éxito de la operación.

		EP 1.2: Cancelar.	- Se selecciona la licencia. - Se presiona el botón **Descargar**. - El sistema muestra una interfaz para que el usuario escoja el destino - Se presiona el botón **Cancelar** para abortar la operación. - El sistema cancela la acción.

1.2 Anexo 2

Acta de liberación de los especialistas del departamento de tecnología del CEIGE.

UCI

CEIGE

DEPARTAMENTO DE TECNOLOGÍA.
martes, 28 de mayo de 2013

A quien pueda interesar:

Por este medio se hace constar que la solución **"Herramienta para la gestión de licencias de software para las soluciones del Centro de Informatización de la Gestión de Entidades (CEIGE)"** del autor **Celso Darien Montero Pérez** fue sometida a dos revisiones técnicas en las cuales se detectaron algunas no conformidades que fueron resueltas en su totalidad quedando esta solución estable y lista para su posterior uso.

Para que así conste firman a continuación los miembros del equipo que realizó las revisiones, el autor y los tutores del trabajo.

Dado a los 28 días del mes de mayo de 2013.

Nombre y apellidos	Firma
Revisores: Magdanis Galván Rey Claudia Bravo Batista	
Autor: Celso Darien Montero Pérez	
Tutores: René R. Bauta Camejo Dausbel Torreblanca Pavó	

Jefe de Departamento de Tecnología
René R. Bauta Camejo

Acta de liberación del cliente.

Acta de aceptación

UCi Universidad de las Ciencias Informáticas

04/06/2013

ACTA DE ACEPTACIÓN

Yo, Johanny Rivera López, como cliente de la solución: Herramienta para la gestión de licencias de software para las soluciones del Centro de Informatización de la Gestión de Entidades, certifico que la aplicación cumple con los requisitos identificados inicialmente y está lista para su uso.

Cliente

Nombre y Apellidos:

Johanny Rivera López

Centro de Informatización de la Gestión de Entidades

UCi

CEIGE

UCi

Universidad de las Ciencias Informáticas

Marco de trabajo para el desarrollo de aplicaciones web de gestión

CENTRO DE INFORMATIZACIÓN DE GESTIÓN DE ENTIDADES

1

GLOSARIO DE TÉRMINOS

[i] **Desoft**: Empresa encargada de ofrecer Soluciones Integrales en Tecnologías de la Información para la Informatización.

[ii] **Transoft**: Software integral para empresas de transporte y logística.

[iii] **CASE :** Ingeniería de Software Asistida por Computadora.

[iv] **BD**: Siglas de Base de Datos.

[v] **Java Swing:** Biblioteca gráfica para Java.

[vi] **ALBET:** Albet Ingeniería y Sistemas, S.A. es una empresa que posee los derechos comerciales de todos los productos y servicios que desarrolla la Universidad de las Ciencias Informáticas, mediante la alianza con otras prestigiosas entidades ofrece soluciones integrales en la esfera de las tecnologías de la información y las comunicaciones.

[vii] **Centro de Soporte:** Entidad encargada del soporte de productos.

[viii] **IEEE 830:** El estándar 830-1998 fue generado por un equipo de trabajo del IEEE, su finalidad es la integración de los requerimientos del sistema desde la perspectiva del usuario, cliente y desarrollador.

[ix] **PDO:** Acrónimo de PHP Data Object, en español Objetos de Datos PHP.

[x] **XML:** Es un lenguaje que ofrece un formato para la descripción de datos estructurados.