



# Sistema para la gestión comercial en el Centro de Gobierno Electrónico.

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

**Autor (es):**

Victor Policarpo Varona Toledo

**Tutor (es):**

Ing. Verónica Hernández Urgelles

Ing. Hernan Antonio Sanchez Guzmán

### **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 firmo la presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

Victor Policarpo Varona Toledo

---

(Autor)

Ing. Verónica Hernández Urgelles

---

(Tutor)

Ing. Hernan Antonio Sanchez Guzmán

---

(Tutor)

## Agradecimientos

**En primer lugar, quiero agradecer a todos los profesores que me han contribuido a lo largo de mi formación profesional, en especial a Dariela que como jefa de año siempre ha estado pendiente de mi situación estudiantil con mucha dedicación. A todos los miembros del tribunal y oponente por haberme ayudado en el desarrollo de la tesis. A mis tutores Hernán y Verónica que fueron los que me instruyeron en todo momento.**

**Quiero agradecerle sobre todo a mi familia, a mi mamá Caridad, mami, a mi papa Victor, papi, a mi hermana Diana, diani y a mi abuela Antonia, mima, que, aunque ya no este entre nosotros sé que debe estar feliz que el niño de sus ojos ya este graduado. De verdad les agradezco luchar a mi lado todos estos años, ustedes no lo hicieron lo mejor posible, lo hicieron que mejor, imposible. Los amo mucho.**

**” Dicen que los amigos son la familia que uno escoge” y la verdad es que yo no puede escoger una mejor familia para acompañarme, por eso les quiero agradecer a mis hermanitos, los que me han soportado tanto. A Falcón, compañero fiel con su verdad absoluta, a Delber con sus tatuajes, a Yeider con sus perfectas combinaciones, a Karel, el pastor, que hizo que me superara siempre y a Osvaldo, pachequin, que ha sido como un tercer tutor para mí. De verdad los quiero.**

**Y cuando de amigos se trata no puedo dejar de agradecerle a todos aquellos con los que también me he alado las orejas, a Omar, machi, a Francis, coquito jr, a Ali, el pana, a Ramón, Ramonnetty, a Luis Rubén, el Rubén, a todos mis compañeros de año y a Camilo, que junto con todos los que vienen de Ciego de Ávila conmigo. De una manera u otra han sido parte también de mi camino. Gracias.**

## Dedicatoria

**Esta tesis va dedicada a mi familia, para que sienta orgullosa del hombre en el que me he convertido, gracias a todos los valores que me inculcaron desde pequeño. En especial a mi Abuela Antonia.**

## Resumen

En el centro de gobierno electrónico, el área relacionada a la gestión comercial de sus proyectos posee un gran flujo de información. Esta se basa esencialmente en las propuestas comerciales de los proyectos, de manera que se generan variados artefactos para su tratado. Estos artefactos son de gran importancia debido a que tienen una relación directa con los clientes. La presente investigación describe el proceso de desarrollo del sistema de gestión comercial para CEGEL, que incrementará la integridad de la información tratada por el asesor de mercadotecnia del centro. Esta herramienta permitirá gestionar toda la información relacionada a las Fichas de costo, Fichas técnicas y Ofertas comerciales del negocio las cuales son fundamentales en esta área. Para lograr este objetivo, se utilizó como metodología de desarrollo el Proceso Unificado Ágil, versión establecida por la Universidad de las Ciencias Informáticas, así como un conjunto de herramientas y lenguajes para el desarrollo de la misma. Las técnicas de obtención de requisitos permitieron definir todos los necesarios para el sistema, y la arquitectura MVC definida por Synfony dio lugar a una correcta implementación. Los resultados fueron validados a través de métricas y pruebas de software, donde se pudo comprobar de forma cuantitativa la calidad del sistema obtenido, así como todas sus funcionalidades.

**Palabras clave:** gestión comercial, ficha de costo, ficha técnica, oferta comercial, integridad de información.

# Índice

Introducción.....	10
<b>Capítulo 1 Fundamentación teórica .....</b>	<b>14</b>
1.1 Introducción.....	14
1.2 Definición de los principales conceptos relacionado con el objeto de estudio y campo de acción.....	14
1.3 Sistemas Homólogos. ....	15
1.3.1 Análisis específico de los sistemas homólogos. ....	16
1.4 Metodología de desarrollo de software. ....	17
1.5 Caracterización de las tecnologías y las herramientas. ....	19
1.5.1 Lenguajes utilizados. ....	19
1.5.2 Sistema gestor de base de datos (SGBD). ....	20
1.5.3 Servidor web.....	20
1.5.4 Herramientas de desarrollo. ....	21
1.5.4 Marco de Trabajo (Framework).....	22
1.6 Conclusiones del capítulo. ....	22
<b>Capítulo 2 Análisis, diseño e implementación del sistema. ....</b>	<b>23</b>
2.1 Introducción.....	23
2.3 Requisitos del software .....	23
2.3.1 Requisitos funcionales .....	23
2.3.2 Requisitos no funcionales .....	26
2.4 Validación de requisitos .....	28
2.5 Historia de usuario .....	30
2.5 Patrones y arquitectura.....	34
2.5.1 Arquitectura del software.....	35
2.5.2 Patrones de diseño .....	36
2.6 Diagrama de clases de diseño .....	39
2.7 Modelo de datos.....	41
2.8 Estándar de codificación.....	43
2.9 Conclusiones de capítulo.....	44
<b>Capítulo 3 Validación del sistema. ....</b>	<b>46</b>
3.1 Introducción.....	46
3.2 Validación del diseño. ....	46
3.2.1 Métrica relaciones entre clases.....	46

<b>3.2.2 Métrica tamaño operacional de clases.....</b>	<b>48</b>
<b>3.3 Pruebas de software.....</b>	<b>49</b>
<b>3.3.1 Pruebas de aceptación.....</b>	<b>50</b>
<b>3.3.2 Pruebas de carga.....</b>	<b>50</b>
<b>3.3.3 Pruebas de caja negra.....</b>	<b>51</b>
<b>3.3.4 Pruebas de caja blanca.....</b>	<b>53</b>
<b>3.4 Conclusiones del capítulo.....</b>	<b>55</b>
<b>Conclusiones generales.....</b>	<b>56</b>
<b>Recomendaciones.....</b>	<b>57</b>
<b>Referencias bibliográficas.....</b>	<b>58</b>

## Índice de Tablas

<i>Tabla 1 Análisis de los sistemas homólogos.....</i>	<i>16</i>
<i>Tabla 2 Requisitos funcionales.....</i>	<i>26</i>
<i>Tabla 3 Historia de usuario (Crear oferta comercial).....</i>	<i>31</i>
<i>Tabla 4 Historia de usuario (Listar oferta comercial).....</i>	<i>32</i>
<i>Tabla 5 Historia de usuario (Mostrar oferta comercial).....</i>	<i>33</i>
<i>Tabla 6 Historia de usuario (Modificar oferta comercial).....</i>	<i>34</i>
<i>Tabla 7 Historia de usuario (Eliminar oferta comercial ).....</i>	<i>34</i>
<i>Tabla 8 Métrica RC.....</i>	<i>47</i>
<i>Tabla 9 Métrica TOC.....</i>	<i>48</i>
<i>Tabla 10 Caso de prueba de aceptación de la HU: Crear oferta comercial.....</i>	<i>50</i>
<i>Tabla 11 Caso de prueba de Caja negra de la HU: Crear oferta comercial.....</i>	<i>52</i>



## Índice de figuras

<i>Figura 1 Diagrama de componente de la arquitectura MVC (Oferta Comercial) .....</i>	<i>36</i>
<i>Figura 2 Uso del patrón experto en la clase Ficha Costo .....</i>	<i>37</i>
<i>Figura 3 Uso del patrón controlador .....</i>	<i>38</i>
<i>Figura 4 Uso del patrón Bajo acoplamiento .....</i>	<i>38</i>
<i>Figura 5 Clase base .....</i>	<i>39</i>
<i>Figura 6 Diagrama de clases del diseño con estereotipos web (Oferta Comercial).....</i>	<i>40</i>
<i>Figura 7 Uso del estándar de codificación LowerCamerCase .....</i>	<i>43</i>
<i>Figura 8 Ejemplo de anotaciones en Ficha Costo.....</i>	<i>44</i>
<i>Figura 9 Resultado de la métrica RC .....</i>	<i>47</i>
<i>Figura 10 Resultados de la métrica TOC .....</i>	<i>49</i>
<i>Figura 11 Total de no conformidades por iteración.....</i>	<i>52</i>
<i>Figura 12 Método newAction() utilizado para la prueba de caja blanca.....</i>	<i>54</i>
<i>Figura 13 Grafo de flujo del método newAction().....</i>	<i>54</i>

## Introducción

La gestión comercial es un término relacionado a la mercadotecnia y se encarga de administrar y gestionar adecuadamente las principales actividades comerciales de la organización en el mercado (Miranda Roque, 2018). Sin dejar de tener en cuenta que la venta es el objetivo final de cualquier empresa desarrolladora de productos.

La Universidad de las Ciencias Informáticas (UCI) no está ajena a este fenómeno, ya que desde su misión se proyecta el desarrollo de productos y servicios que sirvan de soporte a la industria del software en Cuba; los cuales son comercializados por la Dirección de Transferencia Tecnológica (DTT) de la Universidad. En el proceso de comercialización un elemento clave es la gestión de la oferta, en la cual se definen cuestiones como tiempo de desarrollo, costos, recursos humanos, detalles técnicos, alcance del proyecto, beneficios del software y detalles de los servicios a brindar. En este proceso juega un papel determinante el área de mercadotecnia de los centros de desarrollo y se generan como artefactos fundamentales la ficha de costo y una oferta comercial, la cual pasa por varios procesos antes de ser mostrada al cliente, así como el expediente comercial, el cual está conformado por la ficha técnica y el acta de liberación. La Ficha de costo o Base de cálculo es elaborada por el asesor comercial para la definición del costo del producto o servicio y como entrada principal en la elaboración de la oferta comercial.

En el Centro de Gobierno Electrónico (CEGEL), el cual forma parte de la red de centros de la UCI, las informaciones almacenadas en las ofertas comerciales se gestionan manualmente, esto trae como consecuencia la pérdida de información y dificultades para dar seguimiento al proceso. De igual manera, la variedad de formatos y estructura de la documentación que se genera dificulta la estandarización e intercambio de la información. Actualmente existe en la universidad un sistema para la gestión de proyecto conocido por sus siglas como GESPRO, donde el asesor de mercadotecnia sube la propuesta de oferta comercial, así como ficha técnica del producto, para que allí la universidad gestione el negocio y salida del producto, pero este no genera dichas fichas de costo, lo que provoca que el proceso sea engorroso y lento, teniendo en cuenta que se pueden recibir varios proyectos en el mismo mes.

La problemática antes descrita permite identificar el siguiente **problema de investigación**: ¿Cómo gestionar la información asociada al proceso de comercialización de productos y servicios en CEGEL de manera que contribuya a su integridad?

A partir del problema planteado, se obtiene como **objeto de estudio**: gestión comercial de productos y servicios informáticos.

Derivado del objeto de estudio, se plantea como **campo de acción**: sistemas informáticos para la gestión comercial de productos y servicios.

Por lo antes mencionado, se obtiene como **objetivo general**: desarrollar un sistema informático que contribuya a una mejor integridad de la información asociada al proceso de comercialización de los productos y servicios de CEGEL.

El objetivo general se divide en los siguientes **objetivos específicos**:

- Describir el estado del arte referente al desarrollo de herramientas para la gestión de comercial.
- Diseñar las funcionalidades del sistema para la gestión comercial en CEGEL.
- Implementar las funcionalidades del sistema para la gestión comercial en CEGEL.
- Validar el sistema para la gestión comercial en CEGEL aplicando métricas y pruebas de software.

Una vez analizados estos objetivos se obtienen las siguientes **tareas de la investigación**:

- Definición de los principales conceptos relacionado con el objeto de estudio y campo de acción.
- Descripción del proceso de desarrollo de software y su metodología.
- Caracterización de las tecnologías y las herramientas a utilizar para el desarrollo de la solución.
- Especificación de los requisitos funcionales y no funcionales de la solución.
- Descripción de la arquitectura de software base para la implementación del sistema.
- Caracterización y selección de los patrones de diseño más factibles para la propuesta de solución.
- Realización del diagrama de clases de diseño, modelo de datos y diseño de casos de pruebas.
- Implementación de la propuesta de solución.
- Ejecución de las pruebas de software para evaluar la calidad de la implementación.
- Solución de no conformidades al finalizar cada iteración de las pruebas.

Determinándose como **idea a defender**: el desarrollo de un sistema informático que contribuya a una mejor integridad de la información asociada al proceso de comercialización de los productos y servicios de CEGEL.

Durante el desarrollo de la investigación se utilizaron los **métodos científicos** que a continuación se resumen:

#### **Métodos Teóricos:**

- El **Histórico-Lógico**: permitió realizar un estudio sobre las principales tendencias con respecto a los sistemas de gestión comercial ya existentes, tanto en el mundo, como en el marco productivo de la universidad, con el fin de seleccionar las más apropiadas para el desarrollo del sistema.
- El **inductivo-deductivo**: permitió la realización de un estudio más profundo de cada una de las herramientas, lenguajes y metodología a la hora de seleccionar cuales de estos elementos serían utilizados en la investigación y desarrollo del sistema.

#### **Métodos empíricos:**

- La **entrevista**: se utilizó a la hora del análisis del negocio, para así, lograr una visión más amplia del mismo y llegar con mayor eficiencia a los requisitos necesarios para la implementación del sistema.
- La **observación**: se tuvo en cuenta a la hora de valorar la propuesta más adecuada de las tecnologías existentes, además de permitir la observación del proceso de negocio en Centro de Gobierno Electrónico, con el objetivo de conocer su funcionamiento, organización y estructura de la información que contiene.

La tesis consta de la presente introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos que ampliarán la información que se aporta en la investigación.

**Capítulo 1: Fundamentación teórica:** Este capítulo contiene los fundamentos teóricos que sustentan al sistema de gestión de ofertas comerciales. En el mismo se hace un estudio de sistemas homólogos analizando el objetivo, su funcionamiento y tecnologías que utilizan. Además, se describen los principales conceptos para comprender el dominio del problema, así como las tendencias, metodologías, tecnologías y herramientas que se empleará en la implementación de la propuesta de solución.

**Capítulo 2: Análisis, diseño e implementación del sistema.** En este capítulo se explica cómo se desarrolla el flujo actual de los procesos y se describe la propuesta de

solución para resolver el problema planteado. Se establecen los estándares de codificación que se tuvieron en cuenta para el desarrollo del sistema. Por otra parte, se especifican los requisitos funcionales y no funcionales y los elementos fundamentales del diseño y de la arquitectura que se deben tener en cuenta para el desarrollo del sistema, ayudado por la realización de diagramas.

**Capítulo 3: Validación del sistema.** En este capítulo se incluye la programación realizada a partir de los requisitos, teniendo en cuenta las métricas utilizadas para validar el diseño; así como las pruebas de software realizadas para su validación.

## Capítulo 1 Fundamentación teórica

### 1.1 Introducción

En el presente capítulo se define el fundamento teórico de la investigación, exponiéndose los principales conceptos relacionados con el objeto de estudio de la misma, para una mejor comprensión del tema. Se describe además la metodología escogida para guiar el proceso de desarrollo junto al lenguaje de modelado utilizado, el lenguaje de programación, el marco de trabajo a utilizar, entre otras herramientas empleadas. Además, se define la estrategia de prueba a seguir para verificar que la aplicación cumple con los requisitos definidos por el cliente.

### 1.2 Definición de los principales conceptos relacionado con el objeto de estudio y campo de acción.

Con el objetivo de lograr un entendimiento del contenido a tratar en la investigación, se requiere el conocimiento de los conceptos enunciados a continuación:

**Gestión comercial:** la gestión comercial es un término relacionado a la mercadotecnia o marketing, gestionar es hacer posible la realización de una operación comercial. Proporciona las técnicas de como promover un producto o servicio (Borjas & Herrero, 2014).

**Mercadotecnia:** la mercadotecnia o marketing consiste en un conjunto de principios y prácticas que se llevan a cabo con el objetivo de aumentar el comercio, en especial la demanda. El concepto también hace referencia al estudio de los procedimientos y recursos que persiguen dicho fin (Kotler & Fischer, 2015).

**Ficha Técnica del Producto:** la ficha técnica, es un documento en forma de sumario que contiene la descripción de las características técnicas de un objeto, material, producto o bien de manera detallada (Stegfanu, 2016).

**Ficha de costo:** el proceso de planeación reviste especial importancia para las organizaciones, constituyendo un poderoso instrumento de control que influye en la toma de decisiones (Horngren & Blanco, 2015).

**Oferta Comercial:** La propuesta comercial es un documento donde una persona física o jurídica ofrece un producto o servicio a su posible cliente, detallando elementos tales como el precio, plazos de entrega, explicación de características técnicas entre otros (Emprende pyme.net, 2016).

Una vez explicados los principales conceptos se puede dar paso al estudio realizado a distintos sistemas homólogos alrededor del mundo.

### **1.3 Sistemas Homólogos.**

En el estudio realizado a los sistemas existentes de gestión comercial por todo el mundo y dentro del país. Se analizaron varios factores fundamentales, como gestión de la información referente a la Ficha de costo, Ficha técnica, Oferta comercial y licencia de uso. A continuación, se describe todo el proceso de análisis.

#### **Sistema de Gestión Comercial (Insoft)**

Permite la generación automática de facturas a partir de criterios y pedidos con o sin promociones. Para uso de vendedores de ruta. El sistema permite la utilización de computadores Netbooks por vía VPN. Para realizar en línea gestión de cobranzas, captura de pedidos, y facturación. Además, viene con unos módulos de planificación de rutas de venta. Sistema de nómina para todo tipo de empresas comerciales, servicios, manufactureras o industriales. Es una poderosa herramienta de gestión diseñada, a la altura de las necesidades administrativas modernas. El sistema permite que el usuario configure este concepto en el tipo de pedido. Denominado también ventas asistidas. El sistema realiza filtros de búsquedas al momento de realizar el pedido, sugiriendo productos complementarios o sustitutos. Siendo un software de pago desarrollado en Ecuador (InSoft, 2019).

#### **Sistema de gestión Distribución Comercial Mayorista:**

Es un Software ERP integrado, flexible y evolutivo, convirtiéndose en una buena herramienta para gestionar y conectar todos los procesos de negocio. Es una plataforma sectorial, incorpora muchas de las herramientas que las empresas necesitan. Para todas las áreas y departamentos. Desde la gestión de una oferta, la realización de un producto, proyecto o servicio, hasta la tesorería y contabilidad de la empresa. Permite generar y configurar sus ofertas y pedidos, gestionar su seguimiento, reutilizar las ofertas total o parcialmente, generar los pedidos de forma manual y automática. Utiliza la Movilidad para el envío de sus pedidos en su red de negocio. Sistema de Gestión que cuenta con todos los procesos de negocio para la Distribución Comercial Mayorista. No sólo es un software, es una transferencia del conocimiento sectorial. Siendo un software privativo (Sellenne-erp, 2018).

#### **GESTIONPRO**

Permite generar su facturación por ventas en forma fácil, rápida y segura, así como facilitar el control de cuentas corrientes de clientes y proveedores, cheques, bancos y vendedores. Permite obtener más de 190 reportes con información impositiva e histórica por mercadería, servicios, compras y ventas. Moderno. Aplicable para venta de

servicios, distribuidoras, ventas mayoristas y minoristas y productos de fabricación propia. Software de pago desarrollado en Buenos Aires, Argentina (GESTION-PRO, 2018).

### **Sistema de gestión comercial basado en ODOO**

Permite realizar funciones del negocio apoyando la toma de decisiones, gestionar registros internos del proyecto, dar reportes y apoya el estudio del mercado. Desarrollado en software libre y no es privativa. Implementado en la UCI, Cuba.

#### **1.3.1 Análisis específico de los sistemas homólogos.**

Para un mejor análisis de los sistemas estudiados se desarrolla la siguiente tabla:

<b>Sistemas</b>	<b>Ficha Costo</b>	<b>Oferta Comercial</b>	<b>Ficha Técnica</b>	<b>Licencia</b>
Sistema de Gestión Comercial (Insoft)	(NO)	(NO)	(NO)	Privativa
Sistema de gestión Distribución Comercial Mayorista	(NO)	(SI)	(NO)	Privativa
GESTIONPRO	(NO)	(NO)	(NO)	Privativa
Sistema de gestión comercial en ODOO	(NO)	(NO)	(NO)	Libre

*Tabla 1 Análisis de los sistemas homólogos.*

### **Valoración crítica de los sistemas estudiados**

Los sistemas estudiados de gestión de ofertas comerciales, son implementados en software libres y privativos, pero ninguno genera los artefactos necesarios para una correcta gestión, así como tampoco se rigen por las mismas normas y esquemas requeridas por el asesor de mercadotecnia del centro. A partir de lo anterior, se determina que ninguna de las herramientas analizadas responde al problema de la presente investigación, razón por la cual, se decide desarrollar el sistema propuesto.



#### **1.4 Metodología de desarrollo de software.**

Se utiliza la metodología AUP-UCI debido a que es la establecida por el centro.

##### **Proceso Unificado Ágil (AUP)**

AUP es una versión simplificada del Proceso Unificado Racional (RUP). Se describe como una metodología simple y fácil de entender para el desarrollo del software, a su vez utiliza las tendencias y técnicas de RUP. AUP abarca siete flujos de trabajos, cuatro ingenieriles y tres de apoyo: Modelado, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyectos y Ambiente. El modelado agrupa los tres primeros flujos de RUP (Modelado del negocio, Requerimientos y Análisis y Diseño). Dispone de cuatro fases igual que RUP: Creación, Elaboración, Construcción y Transición. Surge por la necesidad de acelerar los proyectos que sean pequeños (Gandarillas, 2016).

La metodología es flexible, está orientada a pequeños equipos y solo se utilizan los artefactos imprescindible y realmente necesarios.

##### **Metodología de desarrollo para la Actividad productiva de la UCI.**

La variación AUP-UCI es una metodología que se puede adaptar a las características de cualquier proyecto de modo que el proceso sea configurable. Entre sus objetivos tiene aumentar la calidad del software que se produce, por ello se apoya en el Modelo CMMI-DEV v1.3 (Díaz Romeu, 2016).

Descripción de las fases **AUP-UCI**:

- **Inicio:** Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
- **Ejecución:** En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
- **Cierre:** En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Descripción de las **disciplinas**:

**Modelado de negocio:** Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Para modelar el negocio se proponen las siguientes variantes: Casos de Uso del Negocio (CUN), Descripción de Proceso de Negocio (DPN) y Modelo Conceptual (MC).

- **Requisitos:** Comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. Existen tres formas de encapsular los requisitos: Casos de Uso del Sistema (CUS), Historias de usuario (HU) y Descripción de requisitos por proceso (DRP).
- **Análisis y diseño:** En esta disciplina los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos. Además, se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales.
- **Implementación:** Se construye el sistema a partir de los resultados del análisis y diseño.
- **Pruebas internas:** Se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas.
- **Pruebas de liberación:** Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.
- **Pruebas de aceptación:** Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

Escenarios para la disciplina Requisitos:

- **Escenario No 1:** Proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS.
- **Escenario No 2:** Proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS.
- **Escenario No 3:** Proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP.
- **Escenario No 4:** Proyectos que no modelen negocio solo pueden modelar el sistema con HU.

El escenario No. 4 se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará

siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información (Gandarillas, 2016).

El sistema en cuestión se desarrollará en el escenario No. 4, ya que se ajusta perfectamente a lo antes descrito.

### **1.5 Caracterización de las tecnologías y las herramientas.**

Se determinaron los siguientes lenguajes, marco de trabajo y herramientas a utilizar para llevar a cabo la implementación del sistema en cuestión.

#### **1.5.1 Lenguajes utilizados.**

A continuación, se detallan los lenguajes utilizados a lo largo del proceso.

##### **HTML 5**

HTML siglas de Hypertext Markup Language es un lenguaje de marcado para la elaboración de documentos web (web pages), define una estructura básica y un código. Cada etiqueta HTML describe diferentes contenidos dentro del documento web, como texto, imágenes, vídeos, entre otros. Es un estándar a cargo de la W3C, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web, sobre todo en lo referente a su escritura e interpretación (HTML5, 2016).

##### **Bootstrap 4.1 (CSS)**

Para la confección del visual se utilizó Bootstrap en su versión 4.1 ya que es un potente framework front-end para el desarrollo de los estilos web, básicamente estructura por clases todo el estilo que se necesite aplicar al contenido web ya desarrollado, siendo la mejor forma de separar el contenido de la página, de su presentación, lo cual mejora la accesibilidad desde disímiles dispositivos y facilita el mantenimiento y perfeccionamiento del contenido (Bootstrap, 2018).

##### **JQuery 1.9**

jQuery es una librería de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web (jQuery, 2019).

##### **PHP 7.2**

Es un lenguaje interpretado de propósito general ampliamente usado, diseñado especialmente para desarrollo web y que puede ser incrustado dentro de código HTML. Es muy rápido, pues permite al equipo de desarrollo la generación dinámica de páginas, su integración con las bases de datos y el servidor Apache. PHP es multiplataforma y

funciona tanto para Unix como para Microsoft Windows, de manera que el código creado para Unix no sufre modificaciones al ser ejecutado en Windows y viceversa. Este lenguaje de programación está preparado para realizar muchos tipos de aplicaciones web gracias a la extensa librería de funciones con la que está dotado (php, 2019).

## **UML**

El Lenguaje de Modelado Unificado (UML:Unified Modeling Language) es la sucesión de una serie de métodos de análisis y diseño orientadas a objetos que aparecen a fines de los 80's y principios de los 90s. Incrementa la capacidad de lo que se puede hacer con otros métodos de análisis y diseño orientados a objetos. Los autores de UML apuntaron también al modelado de sistemas distribuidos y concurrentes para asegurar que el lenguaje maneje adecuadamente estos dominios (Gonzales, 2008).

### **1.5.2 Sistema gestor de base de datos (SGBD).**

Actualmente son muchas las aplicaciones que necesitan acceder a informaciones sin importar la magnitud de los datos. Por tal motivo las aplicaciones necesitan de un medio para acceder a ellos. Es aquí donde aparecen los SGBD, los cuales proporcionan una interfaz entre la aplicación y los datos. A continuación, se describe el SGDB escogido para el desarrollo del sistema.

### **PostgreSQL 9.3**

PostgreSQL es un Sistema de Gestión de Base de Datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones están tan competente como otras bases de datos comerciales. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (PostgreSQL, 2010).

### **1.5.3 Servidor web**

Un servidor web constituye un programa que permite alojar aplicaciones que comúnmente funcionan bajo la filosofía Cliente-Servidor. A continuación, se describe el servidor web escogido para el desarrollo del sistema (Rouse, 2017).

### **Apache 2.4**

Apache 2 es el servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. La licencia Apache es descendiente de la licencia BSD. Esta licencia te

permite hacer lo que quieras con el código fuente. Actualmente más del 60 por ciento de los administradores de toda la Web utilizan Apache. Se trata de la plataforma de servidores Web de código fuente abierto más poderosa del mundo (Rodríguez, 2013).

#### **1.5.4 Herramientas de desarrollo.**

A continuación, se describen las herramientas seleccionadas para la implementación del sistema.

##### **NetBeans IDE 8.2**

Esta es una herramienta de código abierto que permite el desarrollo de aplicaciones con características modulares. Entre sus funcionalidades permite escribir, depurar, compilar y ejecutar programas, aparte de, que soporta otros lenguajes de programación. Se destacan funcionalidades como el auto-completamiento de código, capacidades para el diseño y la integración de Framework de desarrollo. Es un software multiplataforma y extensible que conjuntamente posee abundante documentación referente a las clases que lo conforman (Apache NetBeans, 2018).

##### **Visual Paradigm 8.0**

Herramienta de Ingeniería de Software Asistida por Computadora (CASE) que permite la creación de modelos visuales con Lenguaje de Modelado Unificado (UML). Es una herramienta profesional implementada en lenguaje Java que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Presenta una licencia gratuita y comercial, es de tipo multiplataforma y brinda soporte al ciclo de vida del software. El software de modelado UML ayuda a una rápida 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 (Paradigm, 2016).

##### **PgAdmin 10.3**

Es una plataforma de desarrollo para el gestor de base de datos PostgreSQL. Este software fue diseñado para responder a las necesidades de todos los usuarios, desde la escritura de simples consultas SQL a la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de PostgreSQL y facilita la administración. La conexión al servidor puede hacerse mediante TCP/IP y puede encriptarse mediante SSL para mayor seguridad. Algunas de sus principales características son: (Tools, 2017)

- Multiplataforma.

- Diseñado para múltiples versiones de PostgreSQL y derivados.
- Amplia documentación.
- Acceso a todos los objetos de PostgreSQL.
- Interfaz multilingüe.

#### **1.5.4 Marco de Trabajo (Framework).**

##### **Symfony 3.4**

El marco de trabajo o framework permite simplificar el desarrollo de un sistema mediante la utilización de distintos tipos de lenguajes, el uso de componentes, librerías y estructuras de acuerdo a los diferentes patrones existentes. Aporta agilidad y facilidad a los programadores, pues encapsula operaciones complejas en instrucciones sencillas y posee una organización bien estructurada del proyecto informático en desarrollo. Symfony3.4 es un marco estable para desarrollar aplicaciones, está publicado bajo una licencia de software libre y es fácil de instalar y configurar en la mayoría de las plataformas. Es sencillo de usar y al mismo tiempo es flexible. Su arquitectura interna está completamente desacoplada, lo que permite adaptarse fácilmente a los diferentes proyectos. Posee la ventaja de ser multiplataforma y compatible con la mayoría de gestores de bases de datos como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Hace uso del patrón arquitectónico MVC y sigue la mayoría de las mejores prácticas y patrones de diseño para la web (Symfony, 2019).

#### **1.6 Conclusiones del capítulo.**

El estudio de soluciones existentes permitió definir un conjunto de características para ser incorporadas a la solución desarrollada. El estudio de metodologías y tecnologías para el desarrollo de software, permitió seleccionar las más adecuadas para la implementación de la solución propuesta.

## **Capítulo 2 Análisis, diseño e implementación del sistema.**

### **2.1 Introducción**

En el presente capítulo se describe la propuesta de solución para resolver el problema planteado. Se establecen los estándares de codificación que se tuvieron en cuenta para el desarrollo del sistema. Por otra parte, se especifican los requisitos funcionales y no funcionales y los elementos fundamentales del diseño y de la arquitectura que se deben tener en cuenta para el desarrollo del sistema.

### **2.2 Descripción general de la propuesta de solución**

El sistema de gestión comercial de CEGEL es una herramienta desarrollada para los encargados de llevar a cabo el proceso de negocio de los proyectos en el centro. La herramienta consta de varias funcionalidades listadas a continuación que contribuyen con la integridad documental del proceso.

Gestionar la oferta comercial permite crear, modificar y eliminar una plantilla predefinida con los parámetros que requiere tal oferta. De manera que el especialista solo tenga que introducir los datos pertinentes y la plantilla se genera en formato Word.

Gestionar la ficha técnica permite crear, modificar y eliminar una plantilla predefinida con los parámetros que requiere la ficha en cuestión. De manera que el especialista solo tenga que introducir los datos pertinentes y la plantilla se genera en formato Word.

Gestionar ficha de costos permite crear, modificar y eliminar una plantilla predefinida con los parámetros que requiere la ficha en cuestión. De manera que el especialista solo tenga que introducir los datos pertinentes y la plantilla se genera en formato Excel.

Para un mejor entendimiento de las necesidades del cliente y del sistema en cuestión, fue necesario definir un conjunto de requisitos funcionales y no funcionales

### **2.3 Requisitos del software**

La especificación de requisitos de software (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de requisitos funcionales, que describen el funcionamiento del sistema. También contiene requisitos no funcionales que imponen restricciones en el diseño o la implementación, como, por ejemplo, restricciones en el diseño o estándares de calidad (Pressman, 2010).

#### **2.3.1 Requisitos funcionales**

Como se define en la ingeniería de requisitos, los requisitos funcionales establecen los comportamientos del software (Pressman, 2010).

A través de entrevistas con el cliente, se obtuvieron un total de 38 RF para el desarrollo del sistema, los cuales se listan en la siguiente tabla:

<b>Nº</b>	<b>Nombre</b>	<b>Descripción</b>	<b>Complejidad</b>	<b>Prioridad para el cliente</b>
<b>RF1</b>	Autenticar usuario	Permite al usuario ingresar su usuario y contraseña.	Media	Alta
<b>RF2</b>	Insertar usuario	Permite agregar un usuario al sistema	Media	Alta
<b>RF3</b>	Listar usuario	Permite mostrar todos los usuarios existentes en el sistema.	Media	Alta
<b>RF4</b>	Mostrar usuario	Permite mostrar los datos de un usuario.	Media	Alta
<b>RF5</b>	Editar usuario	Permite modificar un usuario del sistema.	Media	Alta
<b>RF6</b>	Eliminar usuario	Permite eliminar un usuario del sistema.	Media	Alta
<b>RF7</b>	Listar roles	Permite mostrar todos los roles en el sistema.	Media	Alta
<b>RF8</b>	Crear ficha técnica	Permite crear una ficha técnica.	Alta	Alta
<b>RF9</b>	Listar fichas técnicas	Permite mostrar todas las fichas técnicas existentes en el sistema.	Media	Alta
<b>RF10</b>	Mostrar ficha técnica	Permite mostrar los detalles de una ficha técnica.	Media	Alta
<b>RF11</b>	Modificar ficha técnica	Permite realizar modificaciones a una ficha técnica.	Alta	Alta
<b>RF12</b>	Eliminar ficha técnica	Permite eliminar una ficha técnica.	Media	Alta
<b>RF13</b>	Exportar ficha técnica	Permite exportar una ficha técnica a documento Word.	Alta	Alta
<b>RF14</b>	Crear oferta comercial	Permite crear una oferta comercial.	Alta	Alta



<b>RF15</b>	Listar ofertas comerciales	Permite mostrar todas las ofertas comerciales existentes en el sistema.	Media	Alta
<b>RF16</b>	Mostrar oferta comercial	Permite mostrar los detalles de una oferta comercial	Media	Alta
<b>RF17</b>	Modificar oferta comercial	Permite realizar modificaciones en una oferta comercial	Alta	Alta
<b>RF18</b>	Eliminar oferta comercial	Permite eliminar una oferta comercial.	Media	Alta
<b>RF19</b>	Exportar oferta comercial	Permite exportar una oferta comercial a un documento Word	Alta	Alta
<b>RF20</b>	Crear institución	Permite crear una institución en el sistema.	Alta	Alta
<b>RF21</b>	Listar institución	Permite mostrar todas las instituciones existentes en el sistema.	Media	Alta
<b>RF22</b>	Modificar institución	Permite modificar una institución en el sistema.	Media	Alta
<b>RF23</b>	Eliminar institución	Permite eliminar una institución del sistema.	Media	Alta
<b>RF24</b>	Insertar servicio	Permite insertar un servicio al sistema.	Alta	Alta
<b>RF25</b>	Listar servicio	Permite mostrar todos los servicios existentes en el sistema.	Media	Alta
<b>RF26</b>	Mostrar servicio	Permite mostrar los detalles de un servicio existente en el sistema.	Media	Alta
<b>RF27</b>	Modificar servicio	Permite modificar un servicio existente en el sistema.	Media	Alta
<b>RF28</b>	Eliminar servicio	Permite eliminar una servicio existente en el sistema.	Media	Alta

<b>RF29</b>	Crear datos	Permite crearles los datos a un tipo de servicio existente en el sistema.	Alta	Alta
<b>RF30</b>	Listar datos	Permite todos los datos asignados a los servicios.	Alta	Alta
<b>RF31</b>	Editar datos	Permite editar los datos de un servicio.	Alta	Alta
<b>RF32</b>	Eliminar datos	Permite eliminar los datos de un servicio.	Alta	Media
<b>RF33</b>	Crear ficha de costo	Permite crear una ficha de costo.	Alta	Alta
<b>RF34</b>	Listar ficha de costo	Permite mostrar todas las fichas de costo existentes en el sistema.	Media	Alta
<b>RF35</b>	Mostrar ficha de costo	Permite mostrar los detalles de una ficha de costo existente en el sistema.	Media	Alta
<b>RF36</b>	Modificar ficha de costo	Permite modificar una ficha de costo existente en el sistema.	Media	Alta
<b>RF37</b>	Eliminar ficha de costo	Permite eliminar una ficha de costo existente en el sistema.	Media	Alta
<b>RF38</b>	Exportar ficha de costo	Permite exportar la ficha de costo en un documento Excel.	Alta	Alta

*Tabla 2 Requisitos funcionales*

### **2.3.2 Requisitos no funcionales**

Son restricciones de los servicios del sistema, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes y restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (Pressman, 2010).

Así como los requisitos anteriores, también se definieron un conjunto de requisitos no funcionales mostrados a continuación:

- **Requisitos de usabilidad**

- i) **Tipo de usuario final:**

**RNF1.** EL sistema será utilizado por los responsables de los procesos que se llevan a cabo en el área de la mercadotecnia en CEGEL.

- ii) **Tipo de aplicación informática:**

**RNF2.** El sistema constituye una aplicación web enfocada en las necesidades de CEGEL para el área de la mercadotecnia.

- **Finalidad**

**RNF3.** EL sistema está enfocado en la creación de artefactos esenciales y de apoyo para el área de la mercadotecnia en CEGEL.

- **Ambiente**

**RNF4.** El sistema debe presentar una interfaz legible y simple de usar.

**RNF5.** El sistema se desarrollará con java script en su versión 1.8, HTML en su versión 5.0, y CSS en su versión 3.0, todas del lado del cliente.

**RNF6.** El sistema se desarrollará con PHP en su versión 7.2, del lado del servidor

**RNF7.** Se empleará como marco de trabajo Symfony en su versión 3.4, del lado del servidor.

**RNF8.** Se emplearán como marco de trabajo Bootstrap en su versión 4.1 y JQuery en su versión 3.3, del lado del cliente.

**RNF9.** Se emplea como Sistema Gestor de Base de Datos PostgreSQL en su versión 10.3.

**RNF10.** Las computadoras de los clientes no requerirán de muchas prestaciones.

**RNF11.** El idioma de todas las interfaces del sistema es el español.

**RNF12.** En el sistema no existen más de 2 interfaces para lograr una funcionalidad completa.

- **Requisitos de confiabilidad**

**RNF13.** El sistema debe estar disponible al menos durante toda la jornada laboral.

**RNF14.** El sistema presenta campos obligatorios para garantizar la integridad de la información que se introduce por el usuario.

**RNF15.** El sistema no permite la entrada de datos incorrectos.

**RNF16.** Ante el fallo de una funcionalidad del sistema, el resto de las funcionalidades que no dependen de esta deberán seguir funcionando.

- **Soporte**

**RNF17.** El uso del sistema no requiere un tiempo de preparación previa por los usuarios finales para su explotación debido a que es de fácil uso.

**RNF18.** Se debe recibir entrenamiento para la configuración y el mantenimiento del sistema.

- **Restricciones de diseño**

**RNF19.** Para el montaje del sistema se requiere del Sistema Gestor de Bases de Datos PostgreSQL en su versión 10.3

**RNF20.** Las etiquetas de cada funcionalidad y los campos de cada interfaz deben tener títulos asociados a su función.

**RNF21.** El sistema presenta los términos capitalizados, es decir, la primera palabra tendrá su primera letra en mayúscula.

- **Interfaz**

**RNF22.** El sistema presenta una interfaz legible, simple de usar e interactiva.

**RNF23.** La tipografía y colores deben ser estándares en todo el sistema.

- **Seguridad**

**RNF24.** El sistema brinda la posibilidad de establecer permisos sobre acciones, garantizando que solo acceda quien esté autorizado.

**RNF25.** El sistema muestra las funcionalidades de acuerdo a quien esté autenticado en el mismo.

**RNF26.** El sistema debe garantizar la protección ante acciones no autorizadas.

## **2.4 Validación de requisitos**

El resultado del trabajo realizado es una consecuencia de la Ingeniería de Requisitos (especificación del sistema e información relacionada) y es evaluada su calidad en la fase de validación. La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que

el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto (Pressman, 2010).

Para la validación de los requisitos se tuvieron en cuenta las técnicas de:

- **Creación de prototipos:** en esta aproximación a la validación, se muestra un modelo ejecutable del sistema en cuestión a los usuarios finales y clientes. Así, ellos podrán experimentar con este modelo para constatar si cubre sus necesidades reales (Sommerville, 2011).

En el epígrafe 2.5 se hace referencia a las Historias de usuario como producto de trabajo de la disciplina de requisitos, en la misma se detallan los prototipos obtenidos para la propuesta de solución.

- **Revisiones de requisitos:** esta técnica realiza revisiones de cada requisito, comprobando que los mismos no presenten errores e inconsistencias (Sommerville, 2011).

Esta técnica se llevó a cabo por parte de un equipo de personas, validando que la interpretación de cada una de las descripciones no sea ambigua, ni presenten omisiones, evaluando así la calidad de la especificación de cada uno.

Para medir la calidad de la especificación de los requisitos de software se aplicó la métrica Calidad de la especificación (CE). El empleo de esta métrica permite obtener un alto nivel de entendimiento y precisión de los requisitos, para llegar a ello, se debe primeramente calcular el total de requisitos de software como se muestra a continuación:

**Nr:** total de requisitos de software.

**Nf:** cantidad de requisitos funcionales.

**Nnf:** cantidad de requisitos no funcionales.

$$\mathbf{Nr = Nf + Nnf}$$

Sustituyendo los valores en la ecuación, se obtiene:

$$\mathbf{Nr = 38 + 26}$$

$$\mathbf{Nr = 64}$$

Finalmente, para calcular la Especificidad de los Requisitos (ER) se tuvo en cuenta la ausencia de ambigüedad en los mismos de forma tal que se obtuviera una sola interpretación, lo que requiere que cada uno sea descrito utilizando un término único, para ello se realiza la siguiente operación:

**Nui:** número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas.

$$Q1 = Nui / Nr$$

Para sustituir los valores de las variables en la ecuación se tuvo en cuenta que, de los requisitos especificados para el desarrollo de la propuesta de solución, dos de ellos causaron contradicción (RF2 y RF31) en sus interpretaciones. Por tanto, la variable Q1 obtiene el siguiente valor:

$$Q1 = 62 / 64$$

$$Q1 = 0.96$$

Es importante aclarar que mientras más cerca de 1 está el valor de Q1, menor es la ambigüedad. Teniendo en cuenta el resultado anterior, igual a 0.96, se concluye que el 96% de los requisitos se obtuvieron una sola interpretación. Los dos requisitos identificados como ambiguos fueron modificados y validados para garantizar su correcta interpretación, llegando al resultado ideal de Q1=1.

## 2.5 Historia de usuario

Las historias de usuario son descripciones, siempre muy cortas y esquemáticas, que resumen la necesidad concreta de un usuario al utilizar un producto o servicio, así como la solución que la satisface (Solvingad hoc, 2017).

Se realizaron historias de usuario por cada uno de los requisitos funcionales, exponiendo a continuación las relacionadas a todos los requisitos de oferta comercial.

Historia de usuario	
Numero: 14	<b>Requisito:</b> Crear oferta comercial
Programador: Victor Varona Toledo	<b>Iteración asignada:</b> 1
Prioridad: Alta	<b>Tiempo estimado:</b> 24 horas
Riesgo de desarrollo: N/A	<b>Tiempo real:</b>
Descripción: <ul style="list-style-type: none"> <li>• Permite crear una oferta comercial en el sistema.</li> </ul> Campos: <ul style="list-style-type: none"> <li>• Nombre del proyecto: Campo de entrada de texto con el nombre del proyecto.</li> <li>• Código: Campo que permite escoger el código de fecha a utilizar.</li> <li>• Centro: Campo de entrada de texto con el nombre del centro al q pertenece.</li> <li>• Clasificación: Campo desplegable con las posibles clasificaciones.</li> <li>• Plan: Campo desplegable con los posibles tipos de plan.</li> </ul>	

- Identificador: Campo de texto con palabras claves que sirvan de identificador.
  - Fecha de redacción: Campo que despliega calendario.
  - Fecha de inicio prevista: Campo que despliega calendario.
  - Resumen de la oferta: Campo de texto con el resumen de la oferta.
  - Entregables: Campo donde se definen los entregables correspondientes a cada hito.
  - Costo del Proyecto: Campo con distintos valores predefinidos a especificar el costo.
  - Requerimientos tecnológicos: Campo de texto con los requerimientos técnicos.
  - Condiciones para ejecutar el contrato: Campo de texto con las condiciones para ejecutar el contrato.
  - Consideraciones finales: Campo de texto con las consideraciones finales.
- Botones:
- Adicionar: permite crear la oferta comercial.
  - Regresar al listado: permite regresar al listado de ofertas comerciales

Observaciones: N/A

Prototipo elemental de interfaz gráfica de usuario:

Tabla 3 Historia de usuario (Crear oferta comercial)

Historia de usuario	
Numero: 15	<b>Requisito:</b> Listar oferta comercial
Programador: Victor Varona Toledo	<b>Iteración asignada:</b> 1
Prioridad: Alta	<b>Tiempo estimado:</b> 24 horas
Riesgo de desarrollo: N/A	<b>Tiempo real:</b>
Descripción:	

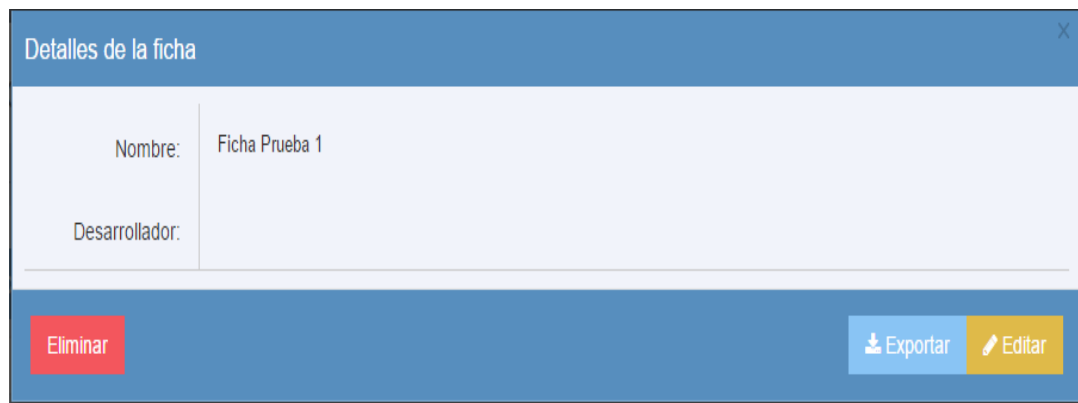
<ul style="list-style-type: none"> <li>Permite listar todas las ofertas comerciales existentes en el sistema.</li> </ul> <p>Botones:</p> <ul style="list-style-type: none"> <li>Nueva oferta: Botón que permite acceder a la vista de creación de ofertas comerciales.</li> <li>Ofertas comerciales: Campos que contienen los detalles de las ofertas comerciales existentes y permiten acceder al mostrar oferta comercial</li> </ul>
Observaciones: N/A
Prototipo elemental de interfaz gráfica de usuario:

*Tabla 4 Historia de usuario (Listar oferta comercial)*

Historia de usuario	
Numero: 17	<b>Requisito:</b> Mostrar oferta comercial
Programador: Victor Varona Toledo	<b>Iteración asignada:</b> 1
Prioridad: Alta	<b>Tiempo estimado:</b> 24 horas
Riesgo de desarrollo: N/A	<b>Tiempo real:</b>
<p>Descripción:</p> <ul style="list-style-type: none"> <li>Permite mostrar los detalles de una oferta comercial</li> </ul> <p>Botones:</p> <ul style="list-style-type: none"> <li>Editar: permite acceder a la edición de la oferta comercial</li> <li>Exportar: permite exportar la oferta comercial a un documento con formato Word</li> </ul>	
Observaciones: N/A	

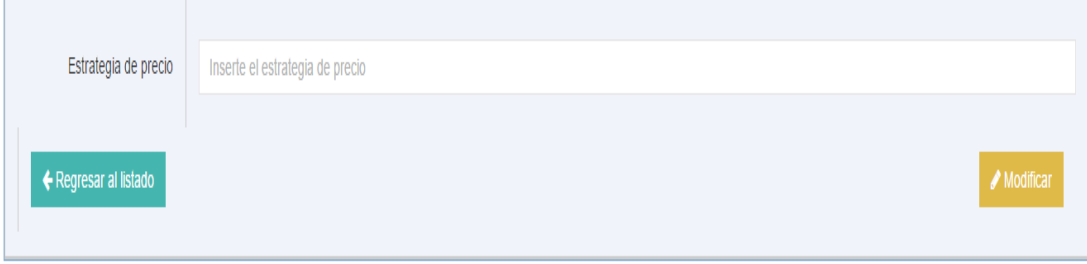


Prototipo elemental de interfaz gráfica de usuario:

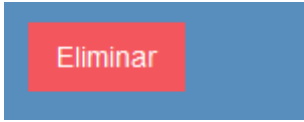


*Tabla 5 Historia de usuario (Mostrar oferta comercial)*

Historia de usuario	
Numero: 18	<b>Requisito:</b> Modificar oferta comercial
Programador: Victor Varona Toledo	<b>Iteración asignada:</b> 1
Prioridad: Alta	<b>Tiempo estimado:</b> 24 horas
Riesgo de desarrollo: N/A	<b>Tiempo real:</b>
<p>Descripción:</p> <ul style="list-style-type: none"> <li>• Funcionalidad que permite modificar una oferta comercial</li> </ul> <p>Campos:</p> <ul style="list-style-type: none"> <li>• Nombre del proyecto: Campo de entrada de texto con el nombre del proyecto.</li> <li>• Código: Campo que permite escoger el código de fecha a utilizar.</li> <li>• Centro: Campo de entrada de texto con el nombre del centro al q pertenece.</li> <li>• Clasificación: Campo desplegable con las posibles clasificaciones.</li> <li>• Plan: Campo desplegable con los posibles tipos de plan.</li> <li>• Identificador: Campo de texto con palabras claves que sirvan de identificador.</li> <li>• Fecha de redacción: Campo que despliega calendario.</li> <li>• Fecha de inicio prevista: Campo que despliega calendario.</li> <li>• Plan de hitos: Campo con distintos valores predefinidos a especificar la duración.</li> <li>• Entregables: Campo donde se definen los entregables correspondientes a cada hito.</li> <li>• Costo del Proyecto: Campo con distintos valores predefinidos a especificar el costo.</li> <li>• Requerimientos tecnológicos: Campo de texto con los requerimientos técnicos.</li> </ul>	

<ul style="list-style-type: none"> <li>• Condiciones para ejecutar el contrato: Campo de texto con las condiciones para ejecutar el contrato.</li> <li>• Consideraciones finales: Campo de texto con las consideraciones finales.</li> </ul> <p>Botones:</p> <ul style="list-style-type: none"> <li>• Modificar: permite guardar los cambios realizados en la oferta en cuestión.</li> <li>• Regresar al listado: permite regresar al listado de ofertas comerciales</li> </ul>
<p>Prototipo elemental de interfaz gráfica de usuario:</p> 

*Tabla 6 Historia de usuario (Modificar oferta comercial)*

Historia de usuario	
Numero: 17	<b>Requisito:</b> Eliminar oferta comercial
Programador: Víctor Varona Toledo	<b>Iteración asignada:</b> 1
Prioridad: Alta	<b>Tiempo estimado:</b> 24 horas
Riesgo de desarrollo: N/A	<b>Tiempo real:</b>
<p>Descripción:</p> <ul style="list-style-type: none"> <li>• Permite eliminar una oferta comercial</li> </ul> <p>Botones:</p> <ul style="list-style-type: none"> <li>• Eliminar: permite eliminar la oferta comercial del sistema</li> </ul>	
Observaciones: N/A	
<p>Prototipo elemental de interfaz gráfica de usuario:</p> 	

*Tabla 7 Historia de usuario (Eliminar oferta comercial)*

## 2.5 Patrones y arquitectura

A continuación, se definirá la arquitectura de software a utilizar para el desarrollo del sistema que no es más que una vista conceptual de toda su estructura y además se determinarán los patrones a aplicar para lograr el esquema organizativo y estructural del software.

### 2.5.1 Arquitectura del software

La arquitectura de software es una disciplina que permite diseñar a un alto nivel la organización de un sistema. La arquitectura de un sistema es a menudo la misma para sistemas con requerimientos similares y, por tanto, pueden soportar reutilización a gran escala (Sommerville, 2011).

Debido a que el lenguaje de programación es PHP, se escogió a Symfony como marco de trabajo para la implementación de las estructuras propuestas en la arquitectura. Señalando así la implementación Modelo-Vista-Controlador (MVC).

En líneas generales, MVC es una propuesta de diseño de software utilizada para implementar sistemas donde se requiere el uso de interfaces de usuario. Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos (Alvarez, 2014).

Esta arquitectura está conformada por tres niveles:

- El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El flujo que sigue el control generalmente es el siguiente:

- El usuario realiza una solicitud a nuestro sistema mediante un navegador web. Generalmente estará desencadenada por acceder a una página. Esa solicitud le llega al controlador.
- El controlador comunica tanto con modelos como con vistas. A los modelos les solicita datos o les manda realizar actualizaciones de los datos. A las vistas les solicita la salida correspondiente, una vez se hayan realizado las operaciones pertinentes según la lógica del negocio.
- Para producir la salida, en ocasiones las vistas pueden solicitar más información a los modelos. En ocasiones, el controlador será el responsable de solicitar todos los datos a los modelos y de enviarlos a las vistas, haciendo de puente entre

unos y otros. Sería corriente tanto una cosa como la otra, todo depende de nuestra implementación; por eso esa flecha la hemos coloreado de otro color.

- Las vistas envían al usuario la salida. Aunque en ocasiones esa salida puede ir de vuelta al controlador y sería éste el que hace el envío al cliente, por eso he puesto la flecha en otro color.

La siguiente figura muestra el diagrama de componentes que explica dicho flujo para la oferta comercial:

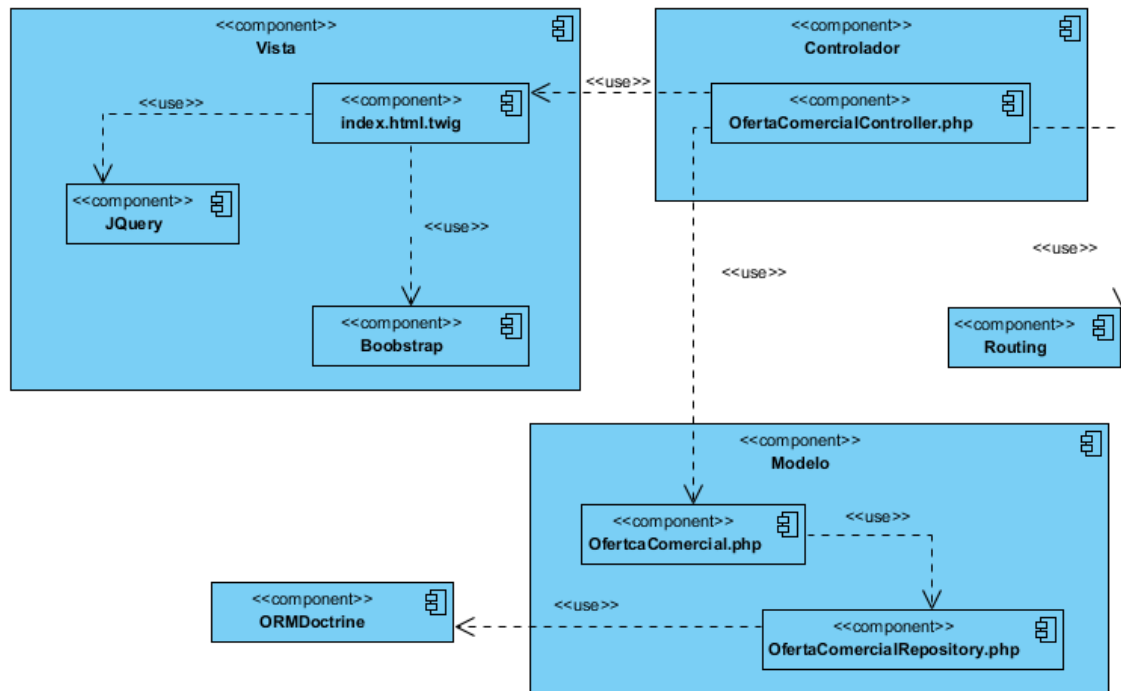


Figura 1 Diagrama de componente de la arquitectura MVC (Oferta Comercial)

## 2.5.2 Patrones de diseño

Los patrones constituyen una guía para el diseño del software. Su objetivo es la solución de problemas que ocurren repetidamente dentro de un contexto muy bien definido. Además, deben ser reusables, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. Son de gran utilidad para describir las mejores prácticas, buenos diseños, y encapsulan la experiencia permitiendo su reutilización. Definen la estructura de un sistema de software, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema. (Sommerville, 2011)

Para la implementación del sistema en cuestión se utilizaron los patrones GRASP y GOF expuestos a continuación.

## Patrones GRASP

**Experto:** propone que la clase que contenga toda la información necesaria será la responsable de la creación de un objeto o la implementación de un método. El comportamiento se distribuye entre las que contienen la información requerida, siendo más fáciles de entender, mantener y ampliar, aumentando sus posibilidades de reutilización (Larman, 2003). Patrón que se utiliza en todas las clases, ya que cada una conoce su información y es la encargada de implementar las funcionalidades que rindan las mismas.

```
3 /** FichaCosto ... */
4 class FichaCosto
5 {
6     /** @var int ... */
7     private $id;
8     /** @var string ... */
9     private $nombre;
10    /** Many Users have Many Groups. ... */
11    private $tipoServicio;
12    /** Many FichaCosto have One Intitucion. ... */
13    private $institucion;
14    /** One FichaCosto has Many OfertaComercial. ... */
15    private $ofertaComercial;
16    /** One FichaCosto has Many Datos. ... */
17    private $datos;
18    /** @return mixed ... */
19    public function getDatos() {...}
20    /** @param mixed $datos ... */
21    public function setDatos($datos) {...}
22    /** @return mixed ... */
23    public function getTipoServicio() {...}
24    /** @param mixed $tipoServicio ... */
25    public function setTipoServicio($tipoServicio) {...}
26    /** Get id ... */
27    public function getId() {...}
28    /** Set nombre ... */
29    public function setNombre($nombre) {...}
30    /** Get nombre ... */
31    public function getNombre() {...}
32    /** Constructor ... */
33    public function __construct() {...}
34    /** Get institucion ... */
35    public function getInstitucion() {...}
36    /** Get ofertaComercial ... */
37    public function getOfertaComercial() {...}
38    /** Add tipoServicio ... */
39    public function addTipoServicio(\AppBundle\Entity\TipoServicio $tipoServicio) {...}
40    /** Remove tipoServicio ... */
41    public function removeTipoServicio(\AppBundle\Entity\TipoServicio $tipoServicio) {...}
42    /** Set institucion ... */
43    public function setInstitucion(\AppBundle\Entity\Institucion $institucion = null)
```

*Figura 2 Uso del patrón experto en la clase Ficha Costo*

**Controlador:** el patrón controlador funciona como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la interfaz quien recibe los

datos del usuario y los envía a las distintas clases según el método invocado (Larman, 2003).

Este patrón fue utilizado en el paquete Controllers que es el encargado de enviar y recibir las instrucciones hacia las demás clases.

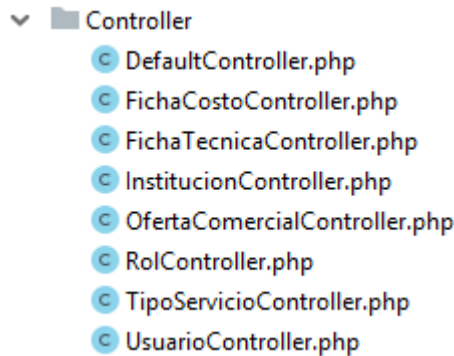


Figura 3 Uso del patrón controlador

**Bajo acoplamiento:** este patrón expresa que entre las clases deberán existir pocas ataduras, es decir, estas estarán lo menos relacionadas posible de forma tal que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, incrementando la reutilización y disminuyendo la dependencia entre las clases (Larman, 2003). Muy poca relación entre clases.

En la figura se muestra la relación de la oferta comercial con la ficha de costo demostrando el bajo acoplamiento. Demostrando que la eliminación de una oferta comercial no afecta las fichas de costo creadas en el sistema.

```
// ...
/**
 * One Product has Many Features.
 * @ORM\OneToMany(targetEntity="OfertaComercial", mappedBy="fichaCosto")
 */
private $ofertaComercial;

// ...
/**
 * Many Features have One Product.
 * @ORM\ManyToOne(targetEntity="FichaCosto", inversedBy="ofertaComercial")
 * @ORM\JoinColumn(name="FichaCosto_id", referencedColumnName="id")
 */
private $fichaCosto;
```

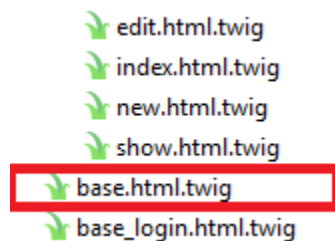
Figura 4 Uso del patrón Bajo acoplamiento

**Alta cohesión:** propone que la información que almacena una clase debe de ser coherente y debe estar, en la medida de lo posible relacionada con la clase. Al realizar un cambio en una clase con alta cohesión, todos los métodos que pueden verse afectados, estarán a la vista, en el mismo archivo. Incrementa la claridad, la reutilización y la facilidad de comprensión del diseño (Larman, 2003).

## Patrones GOF

**Decorador:** permite añadir responsabilidades a objetos concretos de manera dinámica y transparente sin afectar a otros objetos. Este patrón brinda más flexibilidad que la herencia estática y evita que las clases más altas en la jerarquía estén demasiado cargadas de funcionalidad y sean complejas. (Larman, 2003)

Todos los archivos twig de la vista heredaran de base.html.twig y symfony se encarga al renderizar estos archivos de unirlos.



*Figura 5 Clase base*

**Observador:** permite observar los cambios producidos por un objeto, de esta forma, cada cambio que afecte el estado del objeto observado lanzará una notificación a los observadores; a esto se le conoce como Publicador-Suscriptor. Es uno de los principales patrones de diseño utilizados en interfaces gráficas de usuario (GUI), ya que permite desacoplar al componente gráfico de la acción a realizar. (Larman, 2003)

## 2.6 Diagrama de clases de diseño

Un diagrama de clases nos permitirá representar gráficamente y de manera estática la estructura general de un sistema, mostrando cada una de las clases y sus interacciones, representadas en forma de bloques, los cuales son unidos mediante líneas y arcos. Los diagramas de clases son el pilar fundamental del modelado con UML (Cultiracion, 2015). El Diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y las interfaces que participan en el sistema con sus relaciones estructurales y de herencia (Larman, 2003).

Debido a ser un sistema web, se construyó un diagrama de clases con estereotipos web, para poder describir bien el funcionamiento del sistema, como se muestra en la siguiente figura:

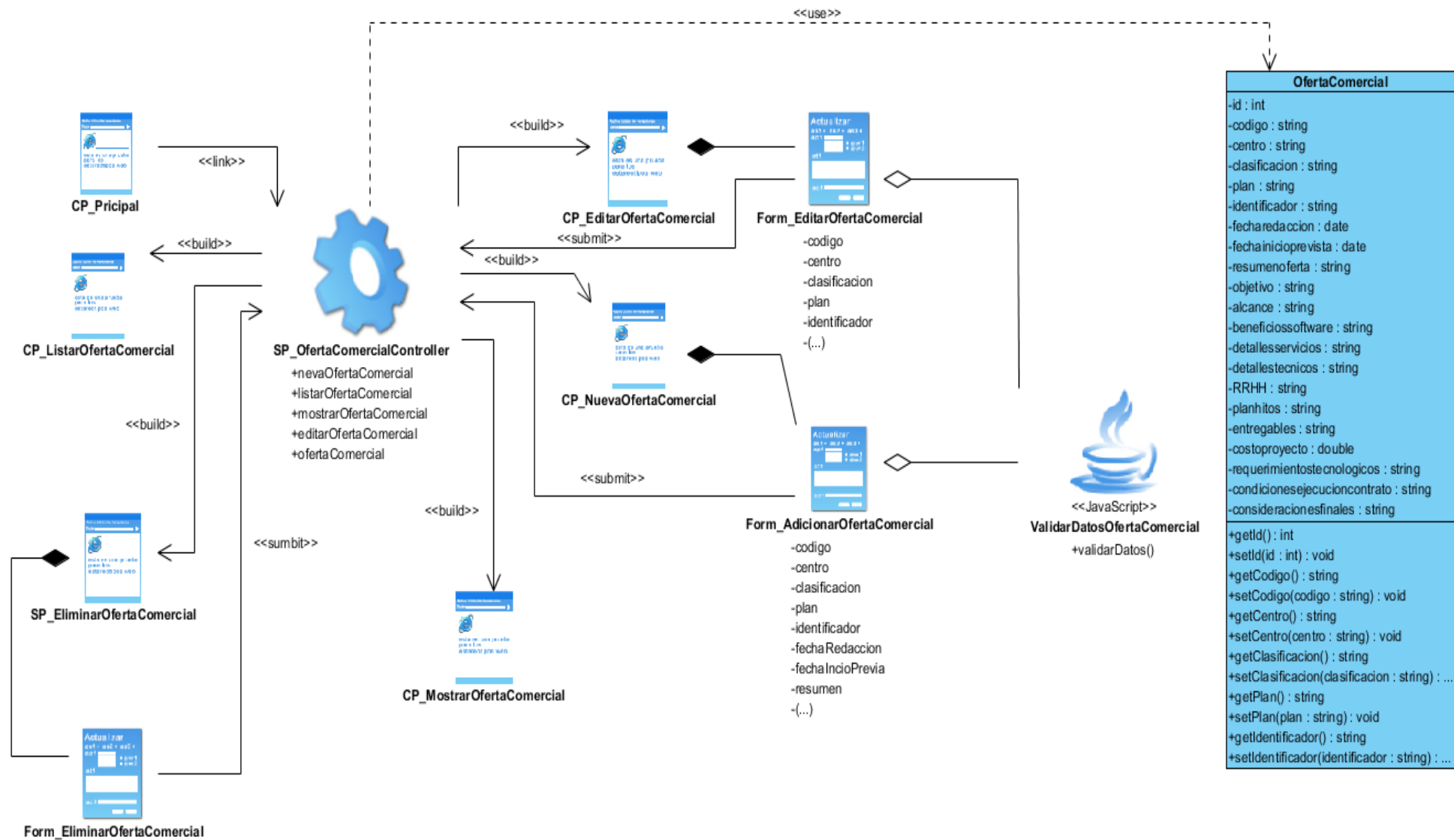


Figura 6 Diagrama de clases del diseño con estereotipos web (Oferta Comercial)



## **2.7 Modelo de datos**

El modelo de datos se centra en el planeamiento del desarrollo de aplicaciones y la decisión de cómo se almacenarán los datos y cómo se accederá a ellos en la base de datos (Pérez Porto & Gardey, Definicion.de, 2008).

Muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se accede a ellos. Los modelos de bases de datos individuales se diseñan en base a las reglas y los conceptos de cualquier modelo de datos más amplio que los diseñadores adopten (Lucidchart, 2016).

Modelo relacional: ordena los datos en tablas, también conocidas como relaciones, cada una de las cuales se compone de columnas y filas. Cada columna enumera un atributo de la entidad en cuestión. En conjunto, a los atributos en una relación se los llama dominio. Se elige un atributo particular o combinación de atributos como clave primaria, a la cual se puede hacer referencia en otras tablas, en donde se la denomina clave externa o llave foránea. Cada fila, también denominada tupla, incluye datos sobre una instancia específica de la entidad en cuestión. El modelo también representa los tipos de relaciones entre esas tablas, incluidas las relaciones uno a uno, uno a muchos y muchos a muchos (Lucidchart, 2016).

Se utiliza este método para el modelado de la base de datos del sistema como se muestra a continuación:

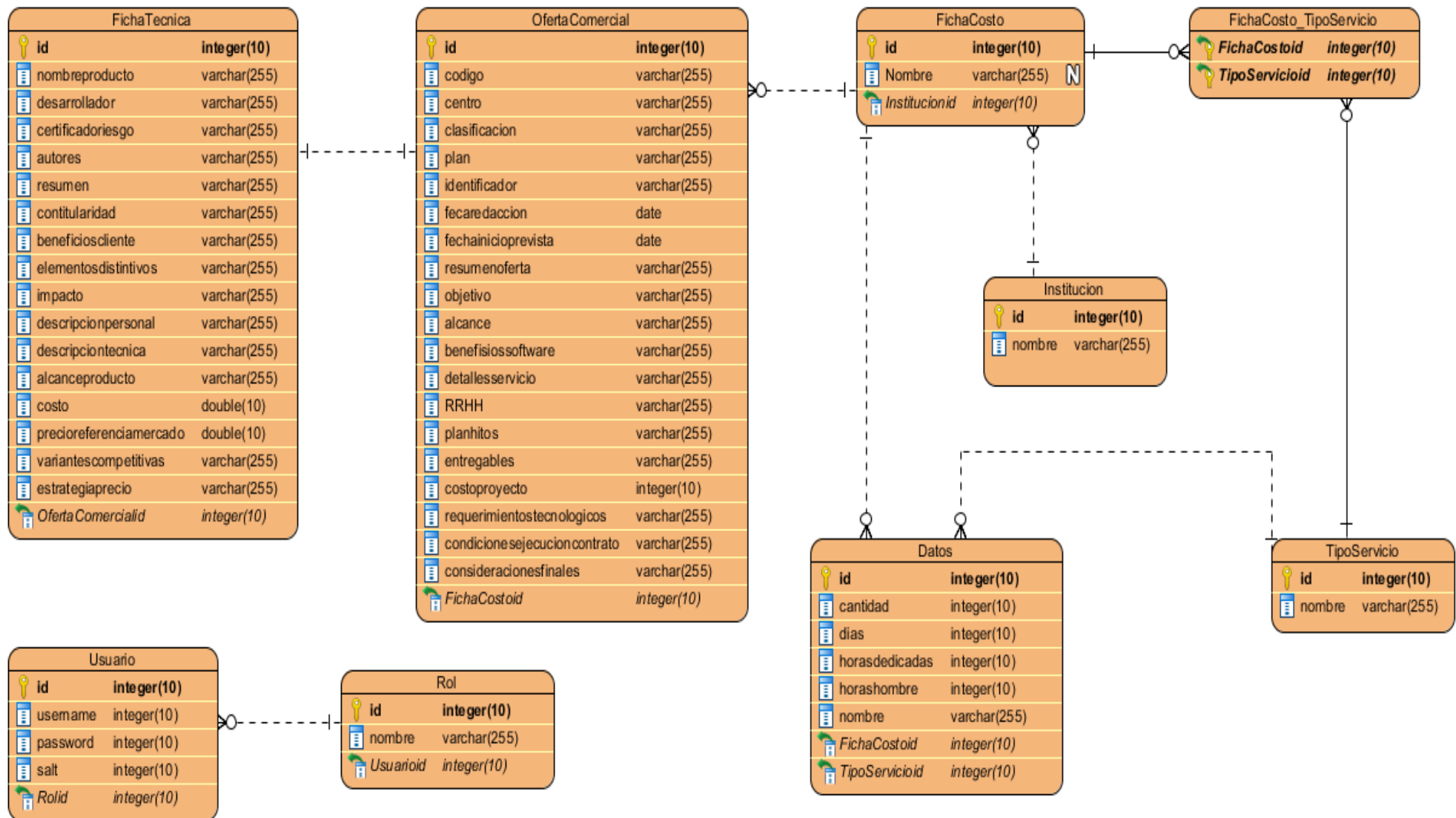


Figura 4 Modelo de datos

## 2.8 Estándar de codificación.

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez (Reyes Muñoz, 2018).

Para una mejor estandarización en el código del sistema se emplea el estándar de codificación CamelCase, y se puede dividir en dos tipos (Acedo, 2017):

- Upper Camel Case, cuando la primera letra de cada una de las palabras es mayúscula
- Lower Camel Case, igual que la anterior con la excepción de que la primera letra es minúscula.

En el caso del sistema en cuestión se utiliza lowerCamelCase, como se muestra en la figura No.7. Lo que hace más sencillo el análisis del código y el mantenimiento del sistema.

```
private $costoProyecto;

/**
 * @var string
 *
 * @ORM\Column(name="requerimientosTecnologicos", type="string")
 */
private $requerimientosTecnologicos;

/**
 * @var string
 *
 * @ORM\Column(name="condicionesEjecucionContrato", type="string")
 */
private $condicionesEjecucionContrato;
```

*Figura 7 Uso del estándar de codificación LowerCamelCase*

## Convenciones de nomenclatura

### General:

- En caso de existir palabras, con la letra ñ, se empleará el término n.
- Para las variables, clases o métodos que contengan tildes serán utilizadas las propias palabras, pero sin la acentuación.
- Serán utilizados nombres relacionados con el contexto en que se desarrolla.

## Indentación

En esta práctica se enfatiza en comenzar a escribir cada línea de código a diferentes distancias desde el borde izquierdo del área de edición. La distancia deberá regirse por la jerarquía que se forma al introducir sentencias dentro de bloques de estructuras. Con el uso de NetBeans IDE los espacios de indentación son ajustados automáticamente, permitiendo a los programadores enfocarse en otras funciones de mayor importancia. Para construir cierta homogeneidad y legibilidad, se escribirá cada sentencia en una línea de código, y en caso de ser necesario cortar las líneas, se hará luego de una coma o antes de un operador.

## Clases:

- El nombre de las clases siempre comenzará con mayúscula. En caso de ser una palabra compuesta, cada una de las palabras comenzarán también de la misma forma.
- Serán utilizados los comentarios en las clases y en las funciones según sea necesario, así como las líneas y espacios en blanco entre funciones y dentro de las mismas llaves.
- Las anotaciones que aplican a una clase, método o constructor aparecen inmediatamente después de la documentación del bloque y cada anotación es listada en una línea propia (es decir, una anotación por línea). Como se muestra en la figura:

```
public function getLevantamientoinformacion()  
{  
    return $this->levantamientoinformacion;  
}
```

*Figura 8 Ejemplo de anotaciones en Ficha Costo*

## Nombre de variables:

- Los nombres de las variables no serán ambiguos.

El correcto uso de cada una de estas nomenclaturas es de gran importancia para el desarrollo del sistema informático debido a que sirven de guía para el entendimiento y estandarización del código del sistema, facilitando su comprensión para la realización de mantenimientos posteriores.

## 2.9 Conclusiones de capítulo.

El empleo de la metodología AUP-UCI en función de describir el desarrollo de la solución propuesta, permitió realizar un trabajo bien estructurado, generando los artefactos

necesarios en cada fase. El uso de la arquitectura MVC permitió diseñar a un alto nivel la organización del sistema, facilitando una guía para la implementación, teniendo en cuenta los patrones de diseño definidos, así como el estándar de implementación lowerCamelCase. Mientras que la especificación de requisitos ayudó a obtener una visión de las funcionalidades al implementar en el sistema propuesto.

## Capítulo 3 Validación del sistema.

### 3.1 Introducción.

En este capítulo se realiza la fase implementación y posteriormente las pruebas al sistema desarrollado. En la fase de implementación corresponde la materialización, en forma de código, de todos los artefactos, descripciones y arquitectura propuesta.

### 3.2 Validación del diseño.

La validación del diseño mediante las métricas de diseño, permite medir de forma cuantitativa la calidad de los atributos internos del software, de forma tal que pueden ayudar al desarrollador a juzgar la calidad de un diseño a nivel de componente (Pressman, 2010).

Para la validación del sistema en cuestión se utilizaron las métricas siguientes:

- Relaciones entre clases (RC)
- Tamaño operacional de clases (TOC)

#### 3.2.1 Métrica relaciones entre clases.

Utilizando la métrica RC el resultado es dado por el número de relaciones de uso que se establecen entre una clase y las demás clases existentes y se evalúa a partir de atributos de calidad; acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas. (Pressman, 2010)

Cada atributo de calidad se describe a continuación:

- **Acoplamiento:** un aumento del RC implica un aumento del acoplamiento de la clase.
- **Complejidad de mantenimiento:** un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- **Reutilización:** un aumento del RC implica una disminución en el grado de reutilización de la clase.
- **Cantidad de pruebas:** un aumento del RC implica un aumento de la cantidad de pruebas necesarias para probar una clase

Para aplicar la métrica RC es necesario categorizar cada una de las clases según la cantidad de relaciones que esta contenga. Para obtener un nivel óptimo de relación entre clases, el valor obtenido al aplicar dicha métrica debe ser directamente proporcional al acoplamiento y a la complejidad de mantenimiento; además debe ser inversamente proporcional al nivel de reutilización del código. A continuación, se describe el atributo de calidad con sus categorías y criterios de evaluación:

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	RC = 0
	Bajo	RC = 1
	Medio	RC = 2
	Alto	RC > 2
Complejidad de Mantenimiento	Baja	RC ≤ Promedio
	Media	Promedio ≤ RC ≤ 2* Promedio
	Alto	RC > 2* Promedio
Reutilización	Baja	RC > 2* Promedio
	Media	Promedio < RC ≤ 2 * Promedio
	Alto	RC ≤ Promedio
Cantidad de pruebas	Baja	RC ≤ Promedio
	Media	Promedio ≤ RC < 2* Promedio
	Alto	RC ≥ 2* Promedio

Tabla 8 Métrica RC

La aplicación del instrumento de evaluación de la métrica RC en el diseño del sistema propuesto, arrojó los resultados expuestos en el gráfico de la figura 8.

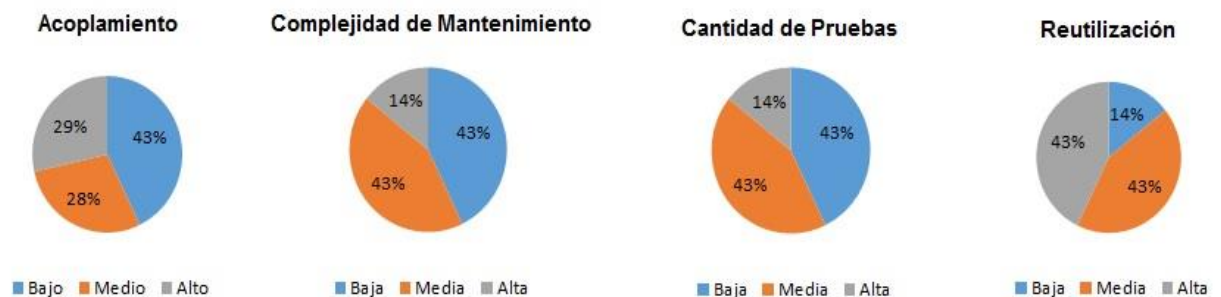


Figura 9 Resultado de la métrica RC

Después de evaluar los resultados obtenidos de cada uno de los atributos de calidad, se obtuvieron los siguientes resultados:

- El 43% de las clases tienen un bajo acoplamiento.
- El 43% de las clases poseen una media complejidad de mantenimiento
- El 43% de las clases tienen una media de cantidad de pruebas
- El 43% de las clases poseen una alta reutilización.

Como se puede observar en la figura el acoplamiento posee un nivel bajo por lo que existe poca dependencia entre las clases trayendo como consecuencia una alta

probabilidad de reutilización. También se puede apreciar que existe un medio nivel de complejidad de mantenimiento por lo que a la hora de optimizar métodos y demás operaciones no es necesario realizar una gran cantidad de pruebas.

### 3.2.2 Métrica tamaño operacional de clases.

Las métricas basadas en el Tamaño Operacional de Clases están dadas por el número de métodos asignados a una clase, así como la cantidad de atributos q esta posea y el promedio que presenta el sistema en su totalidad. Se evalúa a partir de los siguientes atributos de calidad (Pressman, 2010):

- **Responsabilidad:** un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- **Complejidad de implementación:** un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización:** un aumento del TOC implica una disminución del grado de reutilización de la clase.

Una vez analizado el indicador tamaño de clase, si el valor resultante tiende al crecimiento, es probable que la clase posea un alto grado de Responsabilidad; en consecuencia, el nivel de Reutilización sería mínimo y la implementación altamente compleja. A continuación, se describe el atributo de calidad con sus categorías y criterios de evaluación:

Atributo de calidad	Categoría	Criterio
Responsabilidad	Baja	$TOC \leq \text{Promedio}$ (Promedio)
	Media	$TOC \text{ Entre Promedio y } 2^* \text{ Promedio}$
	Alta	$TOC > 2^* \text{ Promedio}$
Complejidad de implementación	Baja	$TOC \leq \text{Promedio}$
	Media	Entre Promedio y $2^* \text{ Promedio}$
	Alta	$TOC > 2^* \text{ Promedio}$
Reutilización	Baja	$TOC > 2^* \text{ Promedio}$
	Media	$TOC \text{ Entre Promedio y } 2^* \text{ Promedio}$
	Alta	$TOC \leq \text{Promedio}$

Tabla 9 Métrica TOC

La aplicación del instrumento de evaluación de la métrica RC en el diseño del sistema propuesto, arrojó los resultados expuestos en el gráfico de la figura 9.





Figura 10 Resultados de la métrica TOC

Después de evaluar los resultados obtenidos de cada uno de los atributos de calidad, se obtuvieron los siguientes resultados:

- El 57% de las clases poseen una baja responsabilidad de prueba.
- El 57 % de las clases poseen una alta reutilización.

Obteniendo un bajo porcentaje de responsabilidad y complejidad de las clases se demuestra la calidad del diseño. Unidos a un alto grado de reutilización de las clases se facilita en gran medida la implementación de las mismas.

### 3.3 Pruebas de software.

Las pruebas de software intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo. El proceso de prueba tiene dos metas definidas: demostrar al desarrollador y al cliente que el software cumple con los requerimientos, y encontrar situaciones donde el comportamiento del software sea incorrecto, indeseable o no esté de acuerdo con su especificación. (Sommerville, 2011)

El sistema paso por las tres etapas de pruebas definidas por (Sommerville, 2011) mostradas a continuación:

- **Pruebas de desarrollo:** donde el sistema se pone a prueba durante el proceso para descubrir errores y defectos.
- **Versiones de pruebas:** donde un equipo de prueba por separado experimenta una versión completa del sistema antes de presentárselo al usuario final.
- **Pruebas de usuario:** donde los usuarios reales o potenciales del sistema prueban al sistema en su propio entorno.

### 3.3.1 Pruebas de aceptación.

La prueba de aceptación es generalmente desarrollada y ejecutada por el cliente en conjunto con el equipo de desarrollo. Esta tiene como objetivo determinar cómo el sistema satisface sus criterios de aceptación, validando los requisitos identificados para el desarrollo del software, incluyendo la documentación y procesos de negocio. Está considerada como la fase final del proceso, para crear un producto confiable y apropiado para su uso (Los Andes Training, 2017).

Para la aplicación de estas pruebas a la solución obtenida, se confeccionaron casos de prueba de aceptación por cada Historia de usuario (HU).

A continuación, se muestra el caso de prueba correspondiente a la HU: crear Oferta Comercial.

<b>Caso de prueba de aceptación</b>		
<b>Código de caso de prueba:</b> 01	<b>Nombre de historia de usuario:</b> Crear oferta comercial	
<b>Nombre de la persona que realiza el caso de prueba:</b> Victor P. Varona Toledo		
<b>Descripción de la prueba:</b> Comprobar que el sistema crea correctamente la oferta comercial, teniendo en cuenta cada condición y los distintos campos rellenos con el formato correspondiente.		
<b>Condiciones de ejecución:</b> El cliente debe acceder al sistema desde cualquier navegador web y una vez accedido hacer un uso correcto de sus privilegios de rol.		
<b>Acción</b>	<b>Entrada</b>	<b>Resultados esperados</b>
Se pulsa el botón adicionar del formulario para su creación.	Texto	Si se crea correctamente la oferta comercial y el sistema muestra el listado de ofertas.
<b>Evaluación de prueba:</b> Satisfactoria		

*Tabla 10 Caso de prueba de aceptación de la HU: Crear oferta comercial*

### 3.3.2 Pruebas de carga.

Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperada. Esta prueba puede mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación, esta prueba puede mostrar el cuello de botella en la aplicación. (Pressman, 2010)

Se realizó la prueba a la gestión de la Oferta comercial, Ficha técnica y la Ficha de costo las cuales representan los procedimientos más complejos del sistema y este se comportó de manera estable.

### 3.3.3 Pruebas de caja negra.

La prueba de caja negra se refiere a las pruebas que se llevan a cabo en la interfaz del software. Una prueba de caja negra examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del software (Globe, 2018).

#### Técnica de prueba: Partición equivalente

Esta técnica divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El método se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de prueba que deben desarrollarse. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada (SCRIBD, 2018).

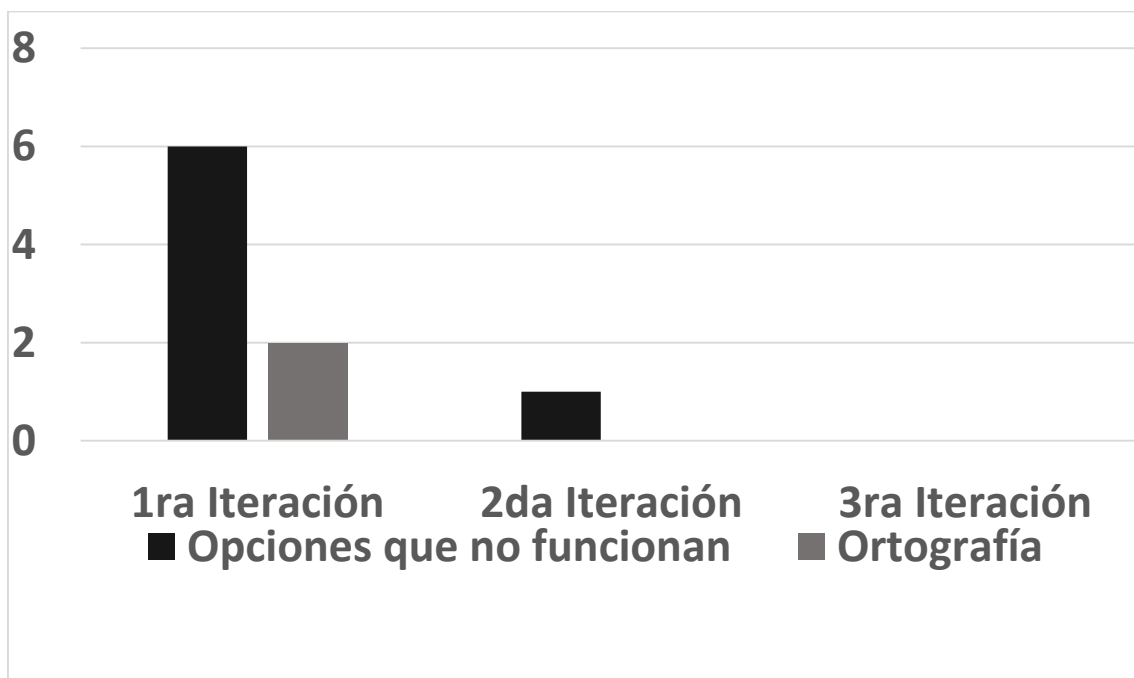
Los objetivos de practicar estas pruebas son: encontrar errores de funciones incorrectas o ausentes, de interfaz o en accesos a la base de datos. A continuación, la tabla 9 muestra el diseño del caso de prueba definido para la historia de usuario: Crear oferta comercial.

Escenario	Descripción	Usuario	Admin	Respuesta del sistema
<b>EC 1.1</b> Crear oferta comercial correctamente.	El usuario introduce los datos correctamente y el sistema permite crear la oferta comercial correctamente.	vpvarona	Si	El sistema crea la oferta comercial satisfactoriamente guardándola en la base de datos para su posterior tratado.
<b>EC 1.2</b> Crear oferta comercial con campos obligatorios en blanco.	El usuario deja campos obligatorios en blanco y el sistema no permite la	vpvarona	Si	El sistema muestra en el campo obligatorio que no se rellenó una notificación "Debe llenar este campo";

	creación de la oferta comercial.			la oferta comercial no se crea.
<b>EC 1.3</b> Crear oferta comercial con datos incorrectos.	El usuario introduce datos incorrectos y el sistema no crea la oferta comercial. (Ejemplo: Números en un campo de texto) .	vpvarona	Si	El sistema muestra en el campo incorrecto una notificación "Valores inválidos", la oferta comercial no se crea .

*Tabla 11 Caso de prueba de Caja negra de la HU: Crear oferta comercial*

Realizándose un caso de prueba por cada requisito funcional, se efectuaron 3 iteraciones para poder alcanzar resultados satisfactorios atendiendo al correcto comportamiento del sistema ante diferentes situaciones. La siguiente figura muestra los factores medidos en cada iteración de la prueba.



*Figura 11 Total de no conformidades por iteración*

Obteniéndose una primera iteración con 6 errores en las funcionales del sistema 2 errores ortográficos. Una segunda iteración con solo 1 error ortográfico y por ultimo una tercera iteración libre de errores. Dando paso así a un sistema libre de errores y sin fallos funcionales.

### 3.3.4 Pruebas de caja blanca.

La prueba de caja blanca del software se basa en el examen cercano de los detalles de procedimiento. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles (TestingBaires, 2018).

Para llevar a cabo las pruebas de caja blanca en el sistema en cuestión se hace uso de la técnica de camino básico.

#### Técnica de camino básico

La técnica del camino básico, consiste en verificar el código de nuestros sistemas de manera que comprobemos que todo funciona correctamente, es decir, se debe verificar que todas las instrucciones del programa se ejecutan por lo menos una vez. (MOUSE, 2017)

Los pasos para desarrollar la prueba del camino básico son:

- Confeccionar el grafo de flujo
- Calcular la complejidad ciclomática
- Determinar el conjunto básico de caminos independientes

Esta métrica se calcula sobre un grafo y se puede realizar mediante tres formas distintas:

1.  $V(G) = R$ .
2.  $V(G) = E - N + 2$
3.  $V(G) = P + 1$

Conociendo que:

- G: Grafo de flujo (grafo)
- R: El número de regiones contribuye a estimar el valor de la complejidad ciclomática.
- E: Número de aristas
- V(G): Complejidad ciclomática
- N: Número de nodos del grafo
- P: Número de nodos predicados incluidos en el grafo.

Una vez calculada la complejidad ciclomática, el valor obtenido representa el límite superior de pruebas que deberán aplicarse.

En la siguiente figura se muestra el método `newAction()` de la clase `OfertaComercialController.php`, perteneciente al requisito funcional crear oferta

comercial. Se selecciona este ya que constituye el método principal de este requisito funcional siendo el que agrega cada nueva oferta comercial al sistema.

```

public function newAction(Request $request)
{
    $ofertaComercial = new OfertaComercial();
    $form = $this->createForm( type: 'AppBundle\Form\OfertaComercialType', $ofertaComercial);
    $form->handleRequest($request);
} ①

if ($form->isSubmitted() && $form->isValid()) { ②
    $em = $this->getDoctrine()->getManager();
    $em->persist($ofertaComercial);
    $em->flush();
    $ofertaComercial = $em->getRepository( className: 'AppBundle:OfertaComercial')->findAll();
    return $this->redirectToRoute( route: 'ofertacomercial_index', array(
        'ofertaComercial' => $ofertaComercial,
    )); ④
}

return $this->render( view: 'ofertacomercial/new.html.twig', array(
    'ofertaComercial' => $ofertaComercial,
    'form' => $form->createView(),
)); ⑤
} ⑥

```

Figura 12 Método newAction() utilizado para la prueba de caja blanca

Obteniéndose el siguiente grafico de flujo:

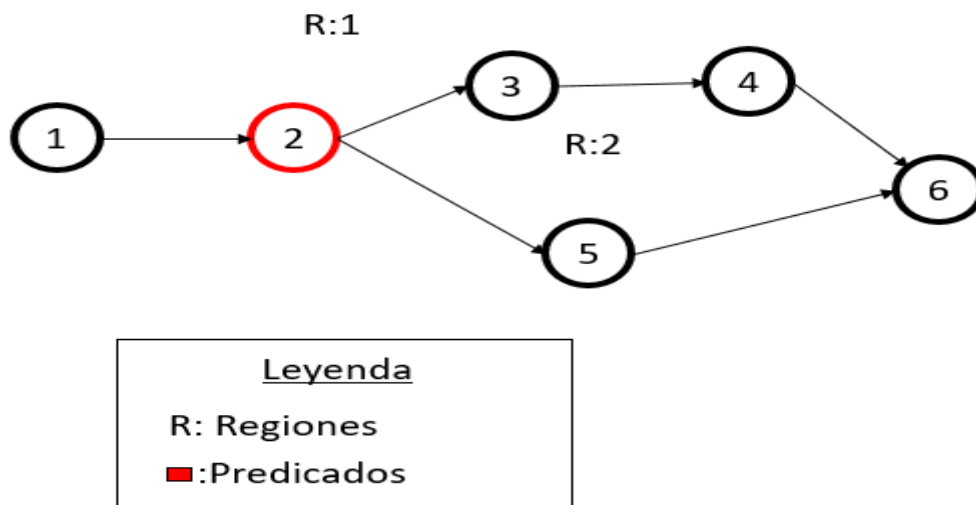


Figura 13 Grafo de flujo del método newAction()

A dicho grafo de flujo se le aplicó la métrica de complejidad ciclomática, calculada por las tres vías conocidas, obteniéndose los siguientes resultados:

- $V(G) = R = 2.$
- $V(G) = 6 - 6 + 2 = 2.$
- $V(G) = 1 + 1 = 2$

Después de calcular la complejidad del grafo, se pudo comprobar que los resultados obtenidos son iguales a 2; por tanto, se deben realizar 2 casos de pruebas, uno por cada ruta independiente. Las rutas independientes resultantes fueron:

- 1- 2- 3- 4- 6
- 1- 2- 5- 6.

### **Resultados al aplicar la técnica de camino básico**

Se aplicó a los métodos de las clases controladoras; ya que estas engloban las funcionalidades del sistema. Para comprobar la fiabilidad del código fuente se realizaron dos iteraciones completas en busca de errores de codificación, el criterio de estimación anteriormente expuesto se tuvo en cuenta como condición de parada. Realizándose así una primera iteración, donde no se obtuvo ninguna no conformidad, por lo que se puede concluir que luego de realizar la prueba de caja blanca, donde se diseñaron y ejecutaron los casos de prueba correspondientes, se logró asegurar el cumplimiento del proceso de mejora del código.

### **3.4 Conclusiones del capítulo.**

A lo largo del capítulo se comprobó el cumplimiento de las funcionalidades del sistema de acuerdo al análisis de las métricas. El correcto uso de las técnicas de validación de diseño RC y TOC permitió comprobar el correcto diseño e implementación del sistema. La aplicación de pruebas en los niveles unidad y aceptación, permitieron obtener un sistema libre de errores que responde a las necesidades del cliente debido a que los resultados obtenidos fueron favorables desde el punto de vista funcional.

## **Conclusiones generales**

El estudio y análisis de los sistemas homólogos existentes, dio como resultado que ninguno genera los artefactos necesarios para una correcta gestión comercial en CEGEL, lo que dio lugar al desarrollo de la investigación realizada. El correcto uso de los métodos científicos permitió identificar las herramientas, lenguajes y metodologías a utilizar; seleccionándose AUP-UCI como metodología de desarrollo, Visual Paradim 8.0 como herramienta para el modelado de los artefactos del análisis y diseño, PostgreSQL 10.3 como sistema gestor de base de datos y NetBeans 8.2 como entorno de desarrollo integrado.

Se identificaron los requisitos funcionales y no funcionales para el sistema, constatándose mediante la aplicación de las métricas de validación de requisitos, que en su totalidad son no ambiguos, comprensibles y correctos. El uso de la arquitectura MVC permitió diseñar a un alto nivel la organización del sistema, facilitando una guía para la implementación, teniendo en cuenta los patrones de diseño definidos, así como el estándar de implementación lowerCamelCase.

Con la implementación se obtuvo el sistema de gestión comercial en CEGEL, respondiendo a todas las necesidades especificadas. Contribuyendo a una integridad óptima de toda la información referente al área comercial del centro. El uso de las técnicas de validación de diseño RC y TOC permitió comprobar el acertado diseño e implementación del sistema, arrojando resultados favorables debido a la existencia de un bajo acoplamiento, alta cohesión en las clases, y alta reutilización por lo que se considera que el diseño es correcto.

La validación de la solución propuesta mediante las pruebas de caja blanca y las pruebas de caja negra permitieron encontrar y corregir los errores no detectados durante la implementación posibilitando cumplir con las especificaciones requeridas y la validación del diseño propuesto, arrojando resultados satisfactorios demostrando la autenticidad de la aplicación desarrollada.



## **Recomendaciones**

Teniendo como base los resultados de este trabajo y la experiencia adquirida durante el desarrollo del mismo, se recomienda:

Realizar un estudio más profundo del mercado internacional para poder obtener nuevas técnicas de gestión comercial.

Ampliar el alcance del sistema a nuevas técnicas de gestión comercial, de manera que pueda adecuarse a estas.

## Referencias bibliográficas

- Acedo, J. I. (1 de Septiembre de 2017). *Apuntes de Programacion*. Obtenido de Estándares de nomenclatura: Snake Case, Kebab Case, Camel Case:  
<http://programacion.jias.es/2017/09/estandares-de-nomenclatura-snake-case-kebab-case-camel-case/>
- Calás, A., Saria, K., & Suarez, R. (2015). *ARQUITECTURA DE REFERENCIA PARA PHP*. Obtenido de  
[https://www.academia.edu/12057345/ARQUITECTURA\\_DE\\_REFERENCIA\\_PARA\\_PHP](https://www.academia.edu/12057345/ARQUITECTURA_DE_REFERENCIA_PARA_PHP)
- Alvarez, M. A. (2 de Enero de 2014). *DesarrolloWeb*. Obtenido de Qué es MVC:  
<https://desarrolloweb.com/articulos/que-es-mvc.html>
- Apache NetBeans*. (2018). Obtenido de Fist the Pieces Together: <https://netbeans.apache.org>
- Bootstrap*. (2018). Obtenido de <https://getbootstrap.com/>
- Borjas, C., & Herrero, J. (2014). *Gestiopolis*. Obtenido de <https://www.gestiopolis.com/que-es-gestion-comercial/>
- Culturacion*. (2015). Obtenido de Diagrama de clases: <http://culturacion.com/que-es-un-diagrama-de-clases/>
- Díaz Romeu, I. (2016). *SISTEMA INFORMÁTICO DE GESTIÓN DE INDICADORES PARA LA FORMACIÓN DE ESTUDIANTES POTENCIALMENTE TALENTOSOS EN LA UCI*. Univercidad de las Ciencias Informaticas, Cuba: Congreso internacional de informacion INFO.
- Emprende pyme.net*. (2016). Obtenido de Propuesta Comercial:  
<https://www.emprendepyme.net/propuesta-comercial.html#bloque-1>
- Gandarillas, A. (2016). *Metodologías*. Obtenido de Ingeniería de Software:  
<https://metodologia.es/aup/>
- GESTION-PRO*. (2018). Obtenido de <http://www.gestionpro.com.ar/>
- Globe*. (2018). Obtenido de Pruebas de caja negra:  
<https://www.globetesting.com/2012/08/pruebas-de-caja-negra/>
- Gonzales, J. E. (2008). *Lenguaje de modelado unificado*. Obtenido de  
<https://www.docirs.com/uml.htm>
- Horngren, C., & Blanco, F. (2015). *Gestionpolis*. Obtenido de  
<https://www.gestiopolis.com/que-es-una-ficha-de-costos/>
- HTML5*. (2016). Obtenido de CAPÍTULO 1 ¿QUÉ ES HTML5?:  
<https://www.arkaitzgarro.com/html5/capitulo-1.html>
- InSoft*. (2019). Obtenido de Informatica y Software Cia Ltda: <http://www.e-insoft.com/web/>
- jQuery*. (2019). Obtenido de write less, do more: <https://jquery.com/>
- Kotler, P., & Fischer, L. (2015). *Promo Negocios*. Obtenido de  
<https://www.promonegocios.net/mercadotecnia/concepto-de-mercadotecnia.html>
- Larman, C. (2003). *UML Y PATRONES. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Segunda edición*. Madrid: PEARSON EDUCACIÓN, S.A.

*Los Andes Training*. (23 de Agosto de 2017). Obtenido de Profesionales del testing: <https://losandestraining.com/2017/08/23/que-son-las-pruebas-de-aceptacion/>

*Lucidchart*. (2016). Obtenido de Que es un modelo de base de datos: <https://www.lucidchart.com/pages/es/que-es-un-modelo-de-datos>

Microsoft Corporation. (2009). *Application architecture guide*. 2009.

Miranda Roque, J. R. (17 de Febrero de 2018). *Gestiopolis*. Obtenido de La Gestión Comercial y la Venta: <https://www.gestiopolis.com/la-gestion-comercial-la-venta/>

MOUSE, J. (2017). *Caja blanca*. Obtenido de Prueba de camino básico: <https://www.jc-mouse.net/ingenieria-de-sistemas/caja-blanca-prueba-del-camino-basico>

MX, E. D. (2016). *Definision MX*. Obtenido de <https://definicion.mx/java/>

Paradigm, V. (2016). *Visual Paradigm*. Obtenido de <https://www.visual-paradigm.com/>

Pérez Porto, J., & Gardey, A. (2008). *Definicion.de*. Obtenido de Modelo de Datos: <https://definicion.de/modelo-de-datos>

Pérez Porto, J., & Gardey, A. (2012). *Definicion.de*. Obtenido de <https://definicion.de/html/>

Pérez Porto, J., & Gardey, A. (2015). *Definición de SQL*. Obtenido de <https://definicion.de/sql/>

*php*. (2019). Obtenido de <https://www.php.net/>

PostgreSQL. (2010). *PostgreSQL*. Obtenido de <https://www.postgresql.org/>

Pressman, R. (2010). *Ingeniería del software. Un enfoque práctico. 6ta Edición*. New York.

Reyes Muñoz, J. R. (2018). *SCRIBD*. Obtenido de Estandares de codificación: <https://es.scribd.com/presentation/289636418/Estandares-de-codificacion>

Rodriguez, K. (2013). *Apache Server*. Obtenido de <http://webapache.blogspot.com/>

Rouse, M. (2017). *TechTarguet*. Obtenido de Servidor Web: <https://searchdatacenter.techtarguet.com/es/definicion/Servidor-Web>

*SCRIBD*. (2018). Obtenido de Casos de Prueba - Particion Equivalente: <https://es.scribd.com/document/348969126/03-Casos-de-Prueba-Particion-Equivalente>

*Selene-erp*. (2018). Obtenido de Tu nueva realidad: <https://www.erp-selene.es/software-erp-especializado/>

*Solvingadhoc*. (Diciembre de 2017). Obtenido de Que son las historias de usuario: <https://solvingadhoc.com/las-historias-usuario-funcion-agilidad/>

Sommerville, I. (2011). *Sommerville\_Parte\_II\_Requerimientos 7ma Edición*. Madrid(España).

Stegfanu, Y. (2016). *Estudios del Mercado*. Obtenido de <http://www.estudiosmercado.com/los-7-puntos-de-una-ficha-tecnica/>

*Symfony*. (2019). Obtenido de SymfonyWorld: <https://symfony.com/roadmap/3.4>

*TestingBaires*. (2018). Obtenido de TESTING DE CAJA BLANCA: <https://testingbaires.com/2014/01/03/testing-de-caja-blanca-parte/>

Tools, p. (2017). *pgAdmin*. Obtenido de pgAdmin Tools: <https://www.pgadmin.org/>