

**Universidad de las Ciencias Informáticas**



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

**Tema:**

**“Herramienta para el análisis de criticidad de activos”**

**Autor:** Yotsan Manuel Hernández Basulto

**Tutores:** Ing. Erich Mario Gómez Pérez  
Ing. Yunior Feria Chapman

**Ciudad de la Habana, Cuba**

**Junio 2016**

**“Año 58 de la Revolución”**

## *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 \_\_\_\_\_.

**Yotsan Manuel Hernández Basulto Erich Mario Gómez Pérez**

\_\_\_\_\_

Firma del Autor    Firma del Tutor

## *Pensamiento*

---



*“ Donde hay una empresa de éxito, alguien tomó alguna vez una decisión valiente.”*

*Peter Drucker*

## *Agradecimientos*

---

*Agradezco infinitamente a mis padres por toda la ayuda que me han brindado siempre, por sus consejos, por el infinito amor que me han dado día a día, y por ser tan especiales conmigo. A mis abuelitos Nieves, Miguelón, María y Evaristo, por ser la fuente de mi inspiración, por ser tan amorosos y comprensivos, por ser como son conmigo, A mis primos y primas Raidel, Disnel, Evelyn, Johander, Dainier, Albertico y en especial a Jahziel que, aunque no se encuentre entre nosotros siempre ocupará un lugar exclusivo en mi corazón. A mis tías y tíos Maité, Yulitza, Rebeca, Marílys, Jorge, Lexmilán, Mantecón, Kíko, Letí, y especialmente a Joel y Juan Miguel, por todos sus consejos, su ayuda y por jugar un papel tan importante en mi vida. Agradezco a mis tutores Erich y Yunion, y a todos mis amigos que de una forma u otra colaboraron en la realización de este trabajo de diploma, fundamentalmente a Aniel, Yaicel, Addiel, Serguey, Asprón y Carlos, a Rolí, Negrín, Araí, Ale y Bauta, por confiar siempre en mí y brindarme su incondicional apoyo. En fin, agradezco a toda mi familia y amigos que siempre me han alentado a seguir estudiando y formándome profesionalmente, y que de una forma u otra aportaron su granito de arena en la elaboración de este trabajo, los quiero con todas mis fuerzas, y no importa cuán distante estemos los unos de los otros, siempre estaremos juntos, porque nos llevamos en el corazón.*

## *Dedicatoria*

---

*Le dedico este trabajo de diploma a mis adorables e incomparables padres Juan Manuel y Misladis, los cuales siempre me han brindado todo su amor, apoyo, dedicación y guía en todos estos años de estudio y formación, permitiendo que todo sea posible sin importar las dificultades y adversidades que la propia vida nos ha impuesto.*

*Los quiero muchísimo, son mi todo.*

## *Resumen*

---

El presente trabajo surge ante la necesidad de viabilizar el proceso de análisis de criticidad en la Ingeniería del Mantenimiento, debido a que actualmente este proceso se lleva a cabo de manera manual. Esta investigación propone la implementación de una herramienta web que permita realizar análisis de criticidades, cuya base esté fundamentada en la automatización de los modelos de criticidad personalizados, validados en las tesis de maestría (García, 2005), (Carballo, 2006), (Pérez, 2008), (Calzada, 2010), (Fernández, 2013), (García, 2015). Con el fin de que se priorice una atención especial a la gama de activos considerados como críticos en las distintas empresas, independientemente de su entorno operacional acorde a las nuevas tendencias del mantenimiento, asegurando de esa manera cambios progresivos y exitosos.

El sistema propuesto como solución brinda una gama de beneficios para los mantenedores, ya que su uso les permitirá realizar análisis de criticidad, así como acciones de inserción, modificación y eliminación sobre los activos. También, podrán consultar un histórico de criticidad donde tendrán la oportunidad de conocer la fecha de realización de dichos análisis, el valor obtenido, y los parámetros que fueron seleccionados en el cálculo.

Para materializar la propuesta planteada, el proceso de desarrollo de software es guiado mediante el uso de la Metodología de desarrollo para la actividad productiva de la UCI, empleando un lenguaje de programación adecuado para el trabajo con fórmulas matemáticas complejas, y un marco de trabajo respaldado por una comunidad de soporte sostenible y con la documentación necesaria para su desarrollo.

**Palabras clave:** activos, análisis, criticidad, proceso.

## Tabla de Contenidos

Introducción .....	- 3 -
<b>CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA .....</b>	<b>- 11 -</b>
<b>1.1 Principales conceptos .....</b>	<b>- 11 -</b>
<b>1.2 Estudio sobre los diferentes modelos de criticidad y herramientas existentes .....</b>	<b>- 12 -</b>
<b>1.3 Metodologías, herramientas y tecnologías de desarrollo .....</b>	<b>- 14 -</b>
1.3.1 Metodología de desarrollo .....	- 14 -
1.3.2 Lenguaje de Modelado .....	- 16 -
1.3.3 Herramienta CASE .....	- 17 -
1.3.4 Lenguaje de Programación .....	- 18 -
1.3.5 IDE de Desarrollo .....	- 19 -
1.3.6 Marco de Trabajo para Servicios Web. ....	- 20 -
1.3.7 Marco de trabajo CSS .....	- 21 -
1.3.8 Marco de Trabajo JavaScript .....	- 23 -
1.3.9 Sistema Gestor de Base de Datos .....	- 24 -
<b>1.4 Conclusiones Parciales .....</b>	<b>- 25 -</b>
<b>Capítulo II: Análisis y diseño de la propuesta de solución .....</b>	<b>- 26 -</b>
<b>2.1 Descripción de la propuesta de solución .....</b>	<b>- 26 -</b>
<b>2.2 Captura de requisitos .....</b>	<b>- 27 -</b>
2.2.1 Requisitos funcionales .....	- 27 -
2.2.2 Requisitos no funcionales .....	- 31 -
2.2.3 Historias de usuarios. ....	- 34 -
2.2.4 Validación de requisitos .....	- 35 -
<b>2.3 Arquitectura del sistema .....</b>	<b>- 35 -</b>
2.3.1 Patrón de Arquitectura .....	- 36 -
<b>2.4 Patrones de diseño .....</b>	<b>- 38 -</b>
2.4.1 Patrones de Diseño de Asignación de Responsabilidades (GRASP) .....	- 38 -
2.4.2 Patrones de Comportamiento (GOF) .....	- 41 -
<b>2.5 Diagrama de clases del diseño con estereotipos web. ....</b>	<b>- 43 -</b>
<b>2.6 Modelo de base de datos .....</b>	<b>- 45 -</b>
2.6.1 Patrones de diseño de la base de datos. ....	- 46 -
<b>CAPÍTULO III: IMPLEMENTACIÓN Y VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN .....</b>	<b>- 48 -</b>

<b>3.1 Implementación</b>	- 48 -
3.1.1 Diagrama de componentes	- 48 -
3.1.2 Diagrama de despliegue	- 50 -
<b>3.2 Estándar de codificación empleado</b>	- 51 -
3.3 Validación	- 52 -
3.3.1 Validación del diseño del sistema	- 52 -
3.4 Pruebas	- 58 -
3.2.1 Pruebas Internas	- 58 -
3.3.2.1 Pruebas de Funcionalidad	- 59 -
3.2.2 Pruebas unitarias	- 62 -
<b>3.3 Caso de estudio</b>	- 64 -
<b>3.4 Conclusiones Parciales</b>	- 67 -
<b>CONCLUSIONES</b>	- 68 -
<b>RECOMENDACIONES</b>	- 69 -
<b>REFERENCIAS BIBLIOGRÁFICAS</b>	- 70 -
<b>ANEXOS</b>	- 74 -





### INTRODUCCIÓN

El hombre que hoy habita la tierra tuvo que, irremediablemente, esforzarse toda su vida en todos los aspectos para resolver problemas que le afectaban, conocer lo inexplicable y mejorar su calidad de vida desde antaño. En la actualidad seguimos inmersos en este proceso infinito de la investigación y el conocimiento, inagotable e inalcanzable a la vez. Tecnologías y diseños de alto nivel ya es cotidiano en cada equipo y sistema en cualquier empresa o negocio pequeño que se analice. Desde la más simple hasta la más potente maquinaria destinada a funcionar, bajo determinadas exigencias o prestaciones, no escapan al latente episodio del paso del tiempo, de los regímenes de operaciones y a los factores del entorno productivo (Pérez, 2008). De ahí el empleo del mantenimiento como una fuente de revitalización funcional de dichas maquinarias que aseguren su reutilización bajo un período de tiempo determinado.

Con el transcurso de los años el mantenimiento ha evolucionado debido a la necesidad de elevar los índices de producción y la calidad de los productos y servicios que se brindan, a la constante aparición de nuevas tecnologías, automatización de los procesos, y con ello al creciente desarrollo de la industria en general. Ha pasado de una actividad reactiva a adoptar una concepción proactiva, dotándole de una visión de negocio. Lo que conlleva a que el ambiente competitivo de las empresas esté caracterizado por una serie de fuerzas que conducen a cambiar la forma tradicional de llevar a cabo sus operaciones y planes de mantenimiento (Díaz, 2011). La rapidez de los cambios en este ambiente de negocio, ha forzado a dichas organizaciones a invertir y tomar decisiones basadas en información incompleta, incierta o imprecisa y al mismo tiempo, cumplir con las exigencias de producir a menor costo y con mayores niveles de calidad.

El mantenimiento se define como el conjunto de actividades que viabilizan la sostenibilidad eficiente y competitiva del ciclo de vida de los activos, procesos, equipos y sistemas, personalizado en su contexto operacional. Tiene como objetivo asegurar la disponibilidad y confiabilidad prevista de las operaciones con respecto a la función deseada. Dando cumplimiento además a todos los requisitos del sistema de gestión de calidad, así como con las normas de seguridad y medio ambiente, buscando el máximo beneficio global (Pérez, 2012). Por tanto, podemos decir que el mantenimiento deberá asegurar la disponibilidad, confiabilidad y seguridad de dichos activos. Para lograrlo deberá estructurarse, organizarse y ejecutarse de manera exacta y meticulosa, acorde a las condiciones específicas de cada lugar, de modo que los resultados del mantenimiento respondan de manera global al cumplimiento de los objetivos de la empresa o institución que se trate.



No todos los activos tienen la misma importancia y el mismo nivel de deterioro sea cual sea la empresa en cuestión y su entorno operacional, ya que unos son más importantes e imprescindibles que otros en el proceso productivo, teniendo en cuenta su aporte económico, social y competitivo. Algunos estarán menos deteriorados que los restantes en dependencia del uso que se le dé y el ambiente en el que opere, además, los recursos en cualquier empresa o institución son limitados, por lo que se debe destinar la mayor parte de dichos recursos a aquellos activos que se consideren más relevantes. Entonces, hacer lo que se debe en el mantenimiento significa asignar correctamente las prioridades, es decir, dirigir los esfuerzos y recursos a las áreas donde sea más importante y/o necesario mejorar la confiabilidad. Donde se define confiabilidad como la probabilidad de que un equipo o sistema opere sin fallas dentro de un período de tiempo bajo unas condiciones de operación previamente establecidas; para así facilitar la toma de decisiones.

Sin embargo, esta asignación de prioridades con frecuencia se hace en base a decisiones tomadas bajo un ambiente de tensión. Entonces, ¿Cómo diferenciar los activos que tienen una gran influencia en los resultados de la empresa de aquellos que no la tienen? Cuando se intenta hacer esa diferenciación podemos decir que se está realizando un Análisis de Criticidad.

Los análisis de criticidades son un instrumento que permiten establecer una jerarquía o prioridades de procesos, sistemas y equipos, creando una estructura que facilita la toma de decisiones, direccionando el esfuerzo y los recursos en áreas donde sea más importante, y así lograr una mejor priorización de los programas y planes de mantenimiento (Mendoza, 2004). Esta priorización comprende desde la programación y ejecución de las órdenes de trabajo, hasta identificar un buen plan de inspección, cuál debe ser el nivel de equipos y piezas de repuesto que deben incluirse en el almacén, así como potenciar el adiestramiento del personal enfocado a las áreas más críticas y la relación costo beneficio.

Estos análisis de criticidades generan una lista ponderada desde el elemento más crítico hasta el menos crítico del total del universo analizado, diferenciando tres zonas de clasificación: alta criticidad, mediana criticidad y baja criticidad. Una vez identificadas estas zonas, es más fácil diseñar una estrategia, para realizar estudios o proyectos que mejoren la toma de decisiones, iniciando las acciones en el conjunto de procesos o elementos que formen parte de la zona de alta criticidad (Mendoza, 2004). El poder disponer de una clasificación jerarquizada de los sistemas y subsistemas permitirá mejorar significativamente la gestión del mantenimiento en las instalaciones, y con ello la toma de decisiones.





La realización de un análisis de criticidad es muy complejo y engorroso, se debe tomar cada activo y definirle un valor por cada indicador. Posteriormente aplicar una fórmula matemática para obtener el resultado final. En un segundo momento se deben listar los activos de mayor criticidad a menor criticidad, calcular la media de criticidad para definir los activos de alta, media y baja criticidad. En organizaciones donde el volumen de activos es significativo, realizar un análisis de criticidad se torna lento y engorroso, y dificulta la toma de decisiones. Actualmente todo este proceso se lleva a cabo de manera manual por los especialistas en mantenimiento, lo cual conlleva a que se consuman varios recursos en las empresas, invirtiendo gran cantidad de tiempo y un sobreesfuerzo por parte de los mantenedores, dando lugar a que se cometan errores de cálculo que evidencien resultados erróneos, pudiendo ocasionar una gran pérdida de presupuesto o inversiones innecesarias.

Otro de los principales problemas existentes es no poder mantener un histórico de los análisis de criticidad para poder realizar comparaciones de la variación de dichos valores en relación a cálculos previos que se hayan llevado a cabo. En algunos contextos operacionales se deben realizar gráficas de criticidad contra complejidad para poder determinar no solo el activo más crítico sino también su complejidad en cuanto a su estructura o acceso para la realización de su mantenimiento.

Por la situación antes expuesta se plantea como **problema de investigación**: ¿Cómo contribuir a mejorar el análisis de criticidad basado en los modelos de criticidad personalizados para una mejor toma de decisiones en la Ingeniería de Mantenimiento?

Se determina como **objeto de estudio** el análisis de criticidad, centrándose específicamente en el **campo de acción**: los modelos de criticidad personalizados.

Para solucionar el problema planteado se define como **objetivo general**: Desarrollar una herramienta informática para el análisis de criticidad basado en los modelos de criticidad personalizados para una mejor toma de decisiones en la Ingeniería de Mantenimiento.

Derivándose los siguientes **objetivos específicos**:

- ✓ Valorar el marco teórico conceptual y el estado del arte respecto a las tecnologías y funcionalidades de los modelos de análisis de criticidad personalizados.
- ✓ Analizar los modelos de criticidad personalizados.
- ✓ Diseñar los componentes por cada uno de los modelos de criticidad personalizados.
- ✓ Implementar los componentes por cada uno de los modelos de criticidad personalizados.



- ✓ Verificar el correcto funcionamiento a través de pruebas de software.
- ✓ Validar la herramienta.

Se plantea como **idea a defender** que el desarrollo de una herramienta informática basado en los modelos de criticidad personalizados contribuirá a mejorar la toma de decisiones en la Ingeniería de Mantenimiento.

Para lograr el objetivo anteriormente planteado se establecen las siguientes **areas de investigación**:

- ✓ Estudio sobre los principales conceptos en el área de conocimiento y los modelos de criticidad validados en tesis de maestría.
- ✓ Selección de las herramientas y tecnologías para el desarrollo.
- ✓ Estudio de la metodología de desarrollo UCI.
- ✓ Identificación de los requisitos funcionales y no funcionales.
- ✓ Definición de los componentes de la herramienta.
- ✓ Implementación de la herramienta.
- ✓ Validación de la herramienta a través de pruebas de funcionalidad, y de aceptación.

Durante la investigación se han empleado un conjunto de **métodos científicos** como procedimientos lógicos, que se han seguido para la obtención y el procesamiento de la información.

Los **métodos de investigación teóricos** que se emplearon fueron:

- ✓ Histórico-Lógico: permitió realizar un estudio sobre los modelos de criticidad, es decir los modelos matemáticos que se emplean para poder llevar a cabo los cálculos de criticidad y complejidad, así como sus parámetros acordes al entorno operacional, con el fin de seleccionar los patrones más apropiados para implementarlos en la propuesta de solución y darle cumplimiento al objetivo general de la presente investigación.
- ✓ Analítico-Sintético: se empleó para realizar el estudio teórico de la investigación, permitiendo el análisis de documentos y la extracción de los elementos más importantes relacionados con los modelos de criticidad personalizados para la realización de los cálculos de criticidad y complejidad de activos
- ✓ Modelación: permitió la creación de una representación de los modelos de criticidad personalizados para realizar cálculos de criticidad y complejidad, haciendo posible el



diseño de los prototipos funcionales y la delegación de los requisitos del sistema informático.



Dentro de los **métodos de investigación empíricos** se utilizaron:

- ✓ Entrevista en Profundidad: se coordinaron varios encuentros con el cliente el cual es Ingeniero en Ciencias Informáticas y está matriculado en la Maestría de Ingeniería y Mantenimiento. Siguiendo como objetivo lograr un mejor entendimiento del negocio a desplegar y una mejor comprensión de sus necesidades, propiciando el levantamiento de los requisitos del sistema y la obtención de información necesaria para el desarrollo de la investigación.
- ✓ Medición: permitió determinar la calidad de la especificación de los requisitos, además de obtener una medida de la calidad del diseño para su validación y el comportamiento de las variables de la investigación.
- ✓ Estudio Bibliográfico: se consultaron disímiles bibliografías referentes al tema de los análisis de criticidad, así como de los modelos de criticidad personalizados.



El cuerpo de la investigación del presente trabajo está estructurado de la siguiente manera:

### **Capítulo 1: Fundamentación teórica**

Este capítulo ofrece los conceptos básicos asociados al dominio del problema, recoge toda la fundamentación teórica que sustenta el desarrollo de la investigación, para la posterior implementación del sistema propuesto como solución. Al mismo tiempo se lleva a cabo un estudio del estado del arte de sistemas existentes con características similares en cuanto a objetivos de uso y de funcionamiento. Se realiza el análisis y selección de la metodología de desarrollo de *software* a emplear en función de las necesidades y las características del grupo de desarrollo. Así como el estudio de las herramientas y tecnologías a emplear en el desarrollo de la solución.

### **Capítulo 2: Análisis y diseño de la propuesta de solución**

En este capítulo se describe las capacidades o funciones que la aplicación debe cumplir (requisitos funcionales) y las propiedades o cualidades que el producto debe tener (requisitos no funcionales). Se hace un análisis de cada uno de los flujos de trabajo, los modelos necesarios para llevar a cabo el desarrollo del sistema. Se realiza una descripción de los casos de uso implementados a partir de las historias de usuario, se realiza el diseño de la aplicación a partir del diagrama de clases de diseño con estereotipos web, y se describe la arquitectura del sistema, así como los patrones de diseño.

### **Capítulo 3: Implementación y validación de la propuesta de solución**

Se abordan aspectos significativos de la implementación del sistema, así como los productos de trabajo generados como parte del proceso de desarrollo. Se realiza un resumen de los resultados obtenidos tras la aplicación de las pruebas de *software* realizadas al sistema con el objetivo de verificar su correcto diseño y funcionamiento en dependencia con las necesidades del cliente.





## **CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA**

En este capítulo se abordan los fundamentos teóricos asociados a la investigación, se enuncian los conceptos relacionados con los modelos de criticidad de activos para la toma de decisiones en la confiabilidad operacional, y la relación existente entre ellos. Se realiza un estudio de diferentes sistemas existentes en Cuba que llevan a cabo estos procesos, y una descripción de las tecnologías utilizadas para el desarrollo de la solución propuesta.

### **1.1 Principales conceptos**

**Análisis de Criticidad:** es una metodología que permite jerarquizar sistemas, instalaciones y equipos, en función de su impacto global, con el fin de facilitar la toma de decisiones. Para realizar un análisis de criticidad se debe definir un alcance y propósito para el análisis, establecer los criterios de evaluación y seleccionar un método de evaluación para jerarquizar la selección de los sistemas objeto del análisis (Mendoza, 2004).

**Confiabilidad:** se define como la probabilidad de que un equipo o sistema opere sin falla por un determinado período de tiempo, bajo condiciones de operación previamente establecidas (Mendoza, 2004).

**Confiabilidad Operacional:** es la capacidad de una instalación o sistema (integrados por procesos, tecnología y personas), para cumplir su función dentro de sus límites de diseño y bajo un contexto operacional específico (Mendoza, 2004).

**Equipos Naturales de Trabajo:** en el contexto de confiabilidad operacional, se define como el conjunto de personas de diferentes funciones de la organización, que trabajan juntas por un período de tiempo determinado en un clima de potenciación de energía, para analizar problemas comunes de los distintos departamentos, apuntando al logro de un objetivo común (Mendoza, 2004).

**Unidades de Proceso:** se define como una agrupación lógica de sistemas que funcionan unidos para suministrar un servicio (Mendoza, 2004).

**Sistema:** conjunto de elementos interrelacionados dentro de las unidades de proceso, que tienen una función específica (Mendoza, 2004).



**Activo:** Un activo es un bien que la empresa posee y que pueden convertirse en dinero u otros medios líquidos equivalentes. Los activos que una empresa posee se clasifican dependiendo de su liquidez, es decir, la facilidad con la que ese activo puede convertirse en dinero. Por ello se dividen en: *Activo fijo* (son los activos utilizados en el negocio y no adquiridos con fines de venta, como maquinarias y bienes inmuebles) y *Activo circulante* (son activos que se esperan que sean utilizados en un periodo inferior al año, como clientes o existencias)(Camarillo, 2016).

**Proceso:** Un proceso es un conjunto de actividades planificadas que implican la participación de un número de personas y de recursos materiales coordinados para conseguir un objetivo previamente identificado. Desde el punto de vista de una empresa, un proceso da cuenta de una serie de acciones que se toman en el aspecto productivo para que la eficiencia sea mayor, en efecto, las empresas buscan continuamente aumentar su rentabilidad produciendo más y bajando sus costos(Justensen, 2015).

## 1.2 Estudio sobre los diferentes modelos de criticidad y herramientas existentes

Para el desarrollo de esta investigación se investigó acerca de herramientas homólogas existentes que hicieran empleo de modelos de criticidad personalizados para calcular criticidad, además de consultó un conjunto de trabajos sobre la temática de análisis de criticidad, desarrollados por personas o empresas de Cuba y otros países de los cuales se referencian los principales a continuación:

### Internacionales:

- ✓ Modelo de Análisis de Criticidad obtenido por la empresa Petróleo de Venezuela SA. (PDVSA)(Mendoza, 2004).
- ✓ Modelo de Criticidad de Subsistemas y Componentes de la Transportadora de Electricidad. SA. (TDE)(Osorio, 2005).
- ✓ Diseño de un Sistema de Mantenimiento con base en Análisis de Criticidad y Análisis de Modos y Efectos de falla en la Planta de Coque de fabricación primaria en la Empresa Acerías Paz del Río S.A. Universidad Pedagógica y Tecnológica de Colombia(Riveros, 2006).
- ✓ Análisis de Criticidad de Plataformas. Activo Integral Cantarell-PEMEX Exploración y Producción. Desarrollo e Implementación de un Modelo de Variables de Estado de Equipos y Estructuras(Martínez, 2008).



**Nacionales:**

- ✓ Modelo de Análisis de Criticidad de los subsistemas objetos de mantenimiento en Instalaciones Hotelera (Gárciga, 2005).
- ✓ Formulación y Validación de una expresión para el análisis de Criticidad del Parque de Equipos de Aeropuertos(Carballo, 2006).
- ✓ Obtención y validación de un modelo para el análisis de criticidad de equipos en plantas de producción de productos biológicos(Pérez, 2008).
- ✓ Análisis de criticidad personalizado de las plantas eléctricas de Grupos Electrógenos de la tecnología fuel oil en Cuba (Calzada, 2010).
- ✓ Obtención y validación de un Modelo de Criticidad para los equipos y sistemas tecnológicos en Plantas de Coque (Fernández, 2013).
- ✓ Obtención y Aplicación de un Modelo de Criticidad para los equipos y sistemas tecnológicos en Plantas Termoeléctricas (García, 2015).

Del estudio y análisis de los modelos de criticidad en plantas y empresas de diferentes países, así como en empresas e instalaciones industriales en Cuba, se puede observar que todas parten de los criterios referenciados en el artículo: El análisis de criticidad, una metodología para mejorar la confiabilidad operacional (Mendoza, 2004), empleando como instrumento en la mayoría de los casos las entrevistas y encuestas a expertos así como las experiencias y vivencias de los especialistas en varias plantas y empresas para determinar las variables o términos de los modelos matemáticos que respondan con mayor exactitud a las características propias y específicas de las mismas.

Luego de haber realizado un análisis de herramientas existentes relacionadas con el tema en cuestión, se encontró una sistema informático que implementa algunos de los modelos de criticidad antes mencionados, la cual fue desarrollada en la Universidad de las Ciencias Informáticas (UCI) sobre el Marco de trabajo Sauxe, la misma actualmente se encuentra en desuso por presentar varias deficiencias tales como: contiene poca documentación para el apoyo en el desarrollo, presenta problemas de rendimiento para realizar consultas de gran cantidad de datos, no cuenta con un histórico de los análisis de criticidad realizados con anterioridad y dejó de recibir soporte debido a la creación de otro marco de trabajo desarrollado sobre Symphony con tecnología más actualizada. Por lo antes analizado, se llegó a la conclusión que sería provechoso utilizar las experiencias positivas del marco de trabajo Sauxe, y se tendrían en cuenta algunas de las funcionalidades implementadas en dicha herramienta, así como una pequeña porción de su diseño.



De lo anteriormente expuesto se deriva la realización de una herramienta informática que desde el punto de vista del negocio integre modelos de criticidad existentes, validados en las tesis de maestría (Gárciga, 2005),(Carballo, 2006), (Pérez, 2008), (Calzada, 2010), (Fernández, 2013),(García, 2015). Empleando de esta manera, herramientas y tecnologías actuales para así asegurar el mayor tiempo de vida útil posible, y facilitar su soporte y mantenimiento. Teniendo en cuenta que a medida que vayan surgiendo nuevos modelos de criticidad y sean validados, se puedan incorporar a esta herramienta. Se hará empleo de un lenguaje de programación que provea las herramientas necesarias para el trabajo con fórmulas matemáticas complejas y un marco de trabajo respaldado por una comunidad de soporte sostenible y con documentación para los desarrolladores.

### **1.3 Metodologías, herramientas y tecnologías de desarrollo**

Elegir correctamente las herramientas, agiliza el proceso de desarrollo de *software*, la selección de éstas se realiza según las necesidades del trabajo a realizar. Luego del estudio realizado por el equipo de desarrollo se identifican cuáles serán las tecnologías y herramientas empleadas para llevar a cabo la solución propuesta. El desarrollo del sistema estará guiado por la metodología de desarrollo de *software* para la actividad productiva de la UCI, Python se empleará como lenguaje de programación, Django como *framework* de desarrollo, Bootstrap como *framework* HTML, CSS y JavaScript, y PyCharm como Entorno Integrado de Desarrollo (IDE). Se decide utilizar la herramienta de modelado Visual Paradigm para los procesos del negocio, la notación BPMN y UML para el resto de las disciplinas y PostgreSQL como Gestor de Base de Datos.

#### **1.3.1 Metodología de desarrollo**

Una metodología de desarrollo de *software* es una guía que muestra paso a paso las tareas y actividades que se deben ejecutar para obtener un producto informático con buena calidad. Indica cuál es el personal que debe participar en el desarrollo de las distintas actividades, así como el papel que deberán enfrentar. Muestra toda la información necesaria para culminar o iniciar alguna actividad que se genere como resultado de los diferentes procesos por los que transcurre la construcción de un *software*(Sommerville, 2005).

Estas se pueden clasificar en dos grupos; metodologías ágiles y metodologías pesadas o robustas. Las metodologías ágiles están orientadas a la interacción con el cliente y el desarrollo incremental del *software*. Muestran al cliente versiones parcialmente funcionales del producto en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios según se desarrolla. Estas



consideran más importante crear una aplicación que funcione, que escribir mucha documentación y la capacidad de respuesta ante un cambio realizado que el seguimiento estricto de un plan (Pérez, 2014). Sin embargo, las metodologías robustas están orientadas a controlar los procesos y establecer de manera rigurosa las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán. Están centradas en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar y requieren una extensa documentación ya que pretenden prever todo de antemano (Pérez, 2014).

### **Metodología de desarrollo para la Actividad productiva de la UCI (AUP-UCI).**

La Universidad de las Ciencias Informáticas tiene entre sus propuestas para guiar el desarrollo de *software* una nueva metodología. Se basa en una variación de la metodología “Proceso Unificado Ágil” (AUP, por sus siglas en inglés) en unión con el modelo CMMI-DEV v1.3. Esta metodología establece distintas fases y disciplinas por las que se debe transitar durante el desarrollo. Las fases definidas son: Inicio, Ejecución (Elaboración, Construcción, Transición) y Cierre. Las disciplinas son: Modelo (Modelado del negocio, Requisitos, Análisis y Diseño), Implementación, Pruebas (Internas, Liberación y Aceptación) y Despliegue (opcional) (Sánchez, 2014).

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (casos de uso del negocio (CUN), descripción de procesos de negocio (DPN) o modelo conceptual (MC)) y existen tres formas de encapsular los requisitos (casos de usos del sistema (CUS), historias de usuario (HU) y descripción de requisitos por proceso (DRP)), surgen cuatro escenarios para modelar el sistema en los proyectos, manteniendo en dos de ellos el MC, quedando de la siguiente forma:

#### **Escenario 1:**

Proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS.

$$\mathbf{CUN + MC = CUS}$$

#### **Escenario 2:**

Proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS.

$$\mathbf{MC = CUS}$$

#### **Escenario 3:**

Proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP.

$$\mathbf{DPN + MC = DRP}$$

#### **Escenario 4:**



Proyectos que no modelen negocio solo pueden modelar el sistema con HU.

### HU

La solución que se desea implementar se enmarca en la fase de *Ejecución*, en la cual se ejecutan las actividades requeridas para desarrollar el *software*. Durante esta fase se realizan los procesos de negocio, se definen los requisitos y se encapsulan a través del Escenario 4 mediante HU, debido a que este escenario 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. En esta fase también se elabora la arquitectura y el diseño, se implementa la solución y se libera el producto (Sánchez, 2014).

Se decide hacer empleo de AUP-UCI teniendo en cuenta que en el trabajo desarrollado con anterioridad (Díaz González, et al., 2014) relacionado con la implementación de una herramienta para llevar a cabo análisis de criticidad, estuvo guiado por el modelo de desarrollo de software que empleaban los proyectos de CEIGE en años anteriores. Dicho modelo se basaba en buenas prácticas y principios de metodologías robustas. En el trabajo mencionado anteriormente se direccionó el mayor esfuerzo a las necesidades y artefactos que eran generados durante el desarrollo de software guiado por dicho modelo. Por ello se concluyó que sería factible en esta investigación, cambiar el enfoque y emplear una metodología que se proyectara de manera preferente hacia una mejor construcción y desarrollo del producto de trabajo, es decir, de la herramienta propuesta como solución, sin dejar de lado los artefactos que se debieran generar. Otros aspectos que contribuyeron a la selección de la metodología de desarrollo AUP-UCI fueron; se contaba con un período de implementación de la herramienta relativamente corto, se disponía de escasos recursos y de un reducido equipo de desarrollo donde el cliente formaba parte del mismo, permitiendo que este también tenga parte de la responsabilidad en el éxito de la construcción del sistema.

#### **1.3.2 Lenguaje de Modelado**

El lenguaje de modelado está formado por un conjunto de símbolos que permitirán modelar parte de un diseño de *software* orientado a objetos. En la informática, como en las demás ramas de la ingeniería se necesita organizar, planificar y diseñar el trabajo antes de ejecutarlo, para así reducir las probabilidades de error en la ejecución del mismo. Los diseños se entienden mejor si se representan gráficamente, por lo que surgió la necesidad de representar los diseños del



*software* que se fueran a desarrollar. Para poder estandarizar estos diseños y que fueran entendibles para todos, surge Lenguaje Unificado de Modelado (UML), dando paso a una de las premisas de la ingeniería del software, la reutilización(Larman, 2003).

### **Lenguaje Unificado de Modelado, UML 2.0**

UML, (por sus siglas en inglés, *Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de *software*, es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables(Romay, 2013).

UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. UML intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos, se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo. Las características más generales de UML 2.0 son: tecnología de orientación a objetos, viabilidad en la corrección de errores, desarrollo incremental e interactivo, participación del cliente en todas las etapas del proyecto. Entre las ventajas que brinda el UML 2.0 se encuentran: mejora el nivel de comunicación formal, el esfuerzo de especificación es más formal, se define, organiza y comparte conocimiento, el impacto de las decisiones sobre un proceso o producto es más visible, el esfuerzo de especificación es más eficiente(Romay, 2013).

### **1.3.3Herramienta CASE**

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE) son aplicaciones informáticas utilizadas para apoyar las actividades del proceso de desarrollo de *software*, con el objetivo de reducir el costo en términos de tiempo y de dinero(William, 1992). Existen variadas herramientas CASE dentro de las que se encuentran Erwin, Rational Rose, Visual Paradigm, las cuales ayudan en las tareas relacionadas con la fase de desarrollo del software como la especificación, estructurado, análisis, diseño, codificación, pruebas y otras actividades como





gestión de proyectos y gestión de la configuración. Estas están destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste en términos de tiempo y de dinero.

### **Visual Paradigm for UML 8.0**

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El *software* de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, estado, entre otros (Barraza, 2014). Su uso permitió crear los diagramas de clases, diagrama de componentes, diagrama de despliegue, realizar código inverso, generar código desde diagramas y generar documentación, es decir, contribuyó en el modelado de los artefactos del análisis y diseño de la solución, así como en una pequeña porción de la implementación de la misma. La herramienta también proporcionó abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Además, identifica requisitos y comunica información, se centra en cómo los componentes del sistema interactúan entre ellos, sin entrar en detalles excesivos, además, permite ver las relaciones entre los componentes del diseño y mejora la comunicación entre los miembros del equipo usando un lenguaje gráfico. Tiene disponible distintas versiones: Enterprise, Professional, Standard, Modeler, Personal y Community. Facilita licencias especiales para fines académicos. Entre sus características más significativas se puede resaltar que su licencia es gratuita, posee varios idiomas, sus ediciones son compatibles y es de fácil uso para la creación de aplicaciones web (Barraza, 2014).

### **1.3.4 Lenguaje de Programación**

Los lenguajes de programación son notaciones para describir la interacción entre las personas y las máquinas. Constituyen el elemento esencial en la construcción de un *software*.

Un lenguaje de programación es un idioma artificial diseñado para controlar el comportamiento de una computadora. Está compuesto por un conjunto de símbolos y reglas; las uniones de estos dos componentes expresan instrucciones que luego serán interpretadas. Los lenguajes de programación pueden clasificarse según el paradigma que usan en: orientado a objetos, funcionales, lógicos y estructurales (Lobos, 2005). Para el desarrollo de esta aplicación se usará un lenguaje orientado a objetos, por la facilidad de uso que estos permiten.





## **Python 2.7**

Python fue creado a finales de la década de los 80 por el holandés Guido van Rossum el cual lanzó en 1991 el código versión 0.9.0. Este lenguaje se puede ejecutar sobre Windows, Linux/Unix, Mac OS X y ha sido portado a Java y .Net máquinas virtuales. Es libre de usar, incluso para productos comerciales, debido a su licencia OSI (Iniciativa de Fuente Abierta) de código abierto. Entre las principales utilidades que brinda este lenguaje de programación se encuentra el ser un lenguaje multiparadigma lo que se puede interpretar como que el mismo permite optar por varios estilos ya sea programación orientada a objetos, estructurada o funcional. Entre las características y ventajas que distinguen a este lenguaje de programación se destacan: su sintaxis es muy clara, simple y sencilla, es un lenguaje de alta potencia, multiplataforma, interpretado, interactivo y orientado a objetos, posee disímiles funciones y gran cantidad de bibliotecas, incorpora el tipado dinámico. Actualmente gana aceptación en el mercado mundial de la industria del *software* por los beneficios que brinda, convirtiéndolo en un fuerte rival para otros lenguajes, asegurando de esta forma su continuidad de uso y actualización con los consiguientes aumentos en soportes y garantías (Alvarez, 2003).

Por tanto, se hace óptimo el uso de Python dada las grandes ventajas que provee, se aprovechó que es un lenguaje de fácil aprendizaje y empleo para cualquier nivel de conocimiento de los desarrolladores, además que su sintaxis es muy clara y sencilla. Se utilizaron varias de sus funcionalidades como el auto completamiento de código, el tipado dinámico, generación de diagramas, así como varias de las bibliotecas que brinda, las cuales lo convierten en un poderoso lenguaje. Durante el desarrollo de la solución se emplearon los paradigmas de programación: Programación Orientada a Objeto (POO), y Programación Funcional. Además, otro de los factores que apoyó su selección como lenguaje de programación a utilizar es que trabaja con entornos de desarrollos con bajo consumo de máquina lo que aumenta el rendimiento de las estaciones de trabajo.

### **1.3.5 IDE de Desarrollo**

Para agilizar la construcción de aplicaciones, los desarrolladores se apoyan de un entorno de desarrollo integrado IDE (Integrated Development Environment). Estos programas contienen un conjunto de herramientas de programación que permiten el desarrollo de otras aplicaciones en determinado lenguaje. Estos pueden realizar las funciones de un editor de código, un compilador, depurador y hasta un constructor de interfaz gráfica (Lau, 2011).



### **JetBrains PyCharm 5.0.4**

Es un IDE inteligente para Python con asistencia y análisis de código para el desarrollo productivo a todos los niveles. Es una herramienta avanzada para el desarrollo profesional en la web y Python. Soporta marcos de trabajo modernos para el desarrollo web, tales como Django, Flask, Google App Engine, Pyramid y web2py. Asimismo, para tecnologías multilenguaje, cubriendo a Python, JavaScript, CoffeeScript, TypeScript, HTML/CSS y Cython. Permite ejecutar, depurar y probar aplicaciones en ordenadores remotos o máquinas virtuales. Brinda soporte para bases de datos con SQL, clases y diagramas de modelos de bases de datos(Villamarin, 2014).

PyCharm busca incrementar la productividad de los desarrolladores dando asistencia en el código, señalamiento de errores, auto-correcciones, así como herramientas de depuración y prueba. Facilita la reestructuración de código fuente de manera automática, así como facilidades para la navegación en el mismo (Elcano, 2014).

Se seleccionó como IDE de desarrollo a PyCharm de la suite de JetBrains por su relación con el lenguaje de programación Python y las facilidades que le brinda al mismo. Este provee auto completamiento inteligente, chequeo de errores, correcciones rápidas y disposiciones para la navegación en el proyecto. Se encarga de todo el trabajo rutinario, enfocándose en elementos más importantes y disfrutar mientras se escribe el código.

### **1.3.6 Marco de Trabajo para Servicios Web.**

En el mundo el desarrollo de aplicaciones informáticas con el uso de los marcos de trabajos se vuelve inminente, ya que estas estructuras conceptuales y tecnológicas de soporte definido, constituyen la base con la cual otro proyecto de *software* puede ser fácilmente organizado y desarrollado(Gudgin, 2007).

Debido a la necesidad de estandarizar las prácticas comunes, así como los criterios y conceptos a la hora de desarrollar aplicaciones de *software*, surgen los marcos de trabajo (en inglés, *frameworks*), que definen una estructura conceptual y tecnológica de soporte definido, a partir de artefactos o módulos de *software* previamente desarrollados, que sirven de base para la organización y construcción de un *software*(Riehle, 2015).



## Django 1.9

Es un *framework* de desarrollo para la web de código abierto desarrollado sobre Python, y cumple con el paradigma del Modelo Vista Controlador (MVC). Inicialmente Django fue desarrollado para gestionar aplicaciones web de páginas orientadas a noticias de World Online, más tarde se liberó bajo licencia BSD. La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, del desarrollo rápido y del principio de DRY (del inglés Don't Repeat Yourself). Python es usado en todas las partes del *framework*, incluso en configuraciones, archivos, y en los modelos de datos(Graham, 2013).

Django promueve el acoplamiento débil, diferentes módulos de la aplicación deberían ser intercambiables y comunicarse con otros módulos a través de APIs limpias y concisas. Este utiliza la arquitectura MVC, es el patrón donde el código para definir y acceder a los datos (modelo) está separado de la lógica de negocio (el controlador) que está separado de la interface de usuario (la vista). A parte de que tiene un único lugar donde guardar la configuración y la capa de acceso a la base de datos tiene un nivel alto de abstracción para poder cambiar el servidor de base de datos (de MySQL a PostgreSQL) de una manera rápida y sencilla (Bennett, 2015).

### Facilidades que ofrece Django:

- ✓ Mapeador objeto-relacional.
- ✓ Interfaz de administración automático.
- ✓ Diseño de URLs elegantes.
- ✓ Sistema de plantillas.
- ✓ Sistema de cache.
- ✓ Internacionalización.

### 1.3.7Marco de trabajo CSS

Genéricamente, un *framework* es un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables. De la misma forma, un *framework* CSS es un conjunto de herramientas, hojas de estilos y buenas prácticas que permiten al diseñador web olvidarse de las tareas repetitivas para centrarse en los elementos únicos de cada diseño en los que puede aportar valor. Los *frameworks* CSS más completos incluyen utilidades para que el diseñador no tenga que trabajar en ningún aspecto genérico del diseño web. Por este motivo, es habitual que los mejores *frameworks* CSS incluyan herramientas para neutralizar los estilos por defecto que aplican los



navegadores, manejar correctamente el texto, de forma que todos los contenidos se vean exactamente igual en todos los navegadores y que sean adaptables para mejorar su accesibilidad y permitir su acceso en cualquier medio y/o dispositivo, y crear cualquier estructura compleja o *layout* de forma sencilla, con la seguridad de que funciona correctamente en cualquier versión de cualquier navegador (Tracey, 2014).

Igual que sus parientes orientados a lenguajes de servidor o cliente, el objetivo de un *framework* CSS será ahorrarnos realizar las tareas básicas al trabajar con hojas de estilo. El uso de un *framework* CSS permite agilizar el desarrollo, sobretodo en sus momentos iniciales, permite ahorra las habituales batallitas entre navegadores para conseguir que tus layouts sean “cross-browser”, partes de una base normalizada / homogeneizada sobre la que desarrollar un trabajo adicional, si el *framework* CSS está bien documentado, agiliza el trabajo en un equipo relativamente grande(McKay, 2011).

### **Bootstrap 3.3.6**

Bootstrap fue creado por Mark Otto y Jacob Thornton para mejorar las herramientas internas en Twitter. Antes se utilizaban muchas librerías diferentes y esto hacía el mantenimiento bastante complicado. En agosto del 2011 Twitter liberó Bootstrap como código abierto (bajo la licencia MIT) y desde febrero del 2012 se convirtió en el proyecto con más forks y favoritos de Github. Bootstrap es un *framework* HTML, CSS y JavaScript que podemos utilizar como base para crear nuestros sitios o aplicaciones web (Wiles, 2014).

El primer beneficio que nos aporta Bootstrap (y cualquier otro *framework*) es el ahorro de tiempo. No tenemos que empezar una página desde cero, sino que podemos pararnos sobre el código que nos aporta y empezar a desarrollar desde ahí. Es fácil de aprender, el sistema de grillas que posee es realmente bueno, posee soporte para los preprocesadores Less y Sass. Está pensado con el diseño móvil primero, con lo cual nuestro sitio va a escalar correctamente sin importar la pantalla que esté utilizando el visitante. Aporta un estilo base a todos los elementos HTML, posee una documentación muy detallada y abundante, cosa que no ocurre con otros *frameworks*. Incluye una lista extensa de componentes que incluye: dropdowns, botones, barras de navegación, alertas, barras de progreso(Wiles, 2014).



### 1.3.8 Marco de Trabajo JavaScript

Un *framework* es una abstracción de código común que provee funcionalidades genéricas que pueden ser utilizadas para desarrollar aplicaciones de manera rápida, fácil, modular y sencilla, ahorrando tiempo y esfuerzo (Pettus, 2014).

En su mayoría, los *frameworks* JavaScript proveen componentes para:

- ✓ **Compatibilidad:** Agregan la posibilidad de escribir código JavaScript totalmente compatible con todos los navegadores y motores JavaScript más utilizados. Esto aumenta la portabilidad y eliminan el “gran dolor de cabeza” de incompatibilidad entre navegadores y sus motores intérpretes JavaScript.
- ✓ **Comunicación asíncrona (Ajax):** Usando este acercamiento, es fácil utilizar XMLHttpRequest para manejar y manipular los datos en los elementos de un sitio bien, aumentando la interactividad y experiencia del usuario.
- ✓ **DOM:** Maximizan la capacidad de agregar, editar, cambiar, eliminar elementos de manera dinámica agregando librerías que facilitan usar DOM.
- ✓ **Validación de Formularios:** Permiten de una manera relativamente fácil validar campos dentro de uno o varios formularios. Esto, desde el punto de vista del desarrollador, simplifica y reduce el código para procesar dichos formularios, ya que los datos llegan previamente validados, reduciendo los errores de tipos de datos.
- ✓ **Efectos visuales:** Utilizando la manipulación de los elementos, se pueden crear efectos visuales y animaciones. Entre los efectos se encuentran: Aparecer y Desaparecer, Redimensionamiento, Move, y más.
- ✓ **Almacenamiento Client-side:** En adición provee funciones para leer y escribir cookies. También proveen una abstracción de almacenamiento que permite a las aplicaciones web guardar datos del lado del cliente, persistente y de manera segura.
- ✓ **Manejo JSON:** Incrementa al máximo el manejo de datos, que pueden ser utilizados para presentar informaciones de manera dinámica y en tiempo de ejecución.
- ✓ **Manejo de Eventos:** Esta característica agregada, permite reaccionar de una manera u otra dependiendo de las acciones del usuario (Pettus, 2014).

### jQuery 1.11.3

jQuery es una librería JavaScript open-source, que funciona en múltiples navegadores, y que es compatible con CSS3. Su objetivo principal es hacer la programación “scripting” mucho más fácil y rápida del lado del cliente. Con jQuery se pueden producir páginas dinámicas, así como animaciones parecidas a Flash en relativamente corto tiempo. La ventaja principal de jQuery es



que es mucho más fácil que sus competidores. Usted puede agregar plugins fácilmente, traduciéndose esto en un ahorro substancial de tiempo y esfuerzo. De hecho, una de las principales razones por la cual Resig y su equipo crearon jQuery fue para ganar tiempo (en el mundo de desarrollo web, tiempo importa mucho). La licencia open source de jQuery permite que la librería siempre cuente con soporte constante y rápido, publicándose actualizaciones de manera constante. La comunidad jQuery es activa y sumamente trabajadora.

Otra ventaja de jQuery sobre sus competidores como Flash y puro CSS es su excelente integración con AJAX. En resumen, jQuery es flexible y rápido para el desarrollo web, viene con licencia MIT y es Open Source, tiene una excelente comunidad de soporte, tiene plugins, bugs son resueltos rápidamente, posee una excelente integración con AJAX(Procida, 2013).

Una de las principales desventajas de jQuery es la gran cantidad de versiones publicadas en el corto tiempo. No importa si usted está corriendo la última versión de jQuery, usted tendrá que hostear la librería usted mismo (y actualizarla constantemente), o descargar la librería desde Google (atractivo, pero puede traer problemas de incompatibilidad con el código). Además del problema de las versiones, otras desventajas que podemos mencionar: jQuery es fácil de instalar y aprender, inicialmente. Pero no es tan fácil si lo comparamos con CSS, Si jQuery es implementado inapropiadamente como un *framework*, el entorno de desarrollo se puede salir de control(Procida, 2013).

### **1.3.9 Sistema Gestor de Base de Datos**

Sistema Gestor de Base de Datos (SGBD) “es un *software* diseñado para asistir al mantenimiento y utilización de extensas colecciones de datos”. Es una aplicación que permite a los usuarios crear, definir y mantener las bases de datos. Un SGBD es una aplicación que facilita la gestión y mantenimiento de forma segura de la información almacenada en las bases de datos. De manera general, los SGBD son un tipo de *software* dedicado a servir de interfaz entre las bases de datos, los usuarios y las aplicaciones que las utilizan, permitiendo así la creación y manipulación de los objetos y las propias bases de datos(Ramakrishnan, 2003).

#### **PostgreSQL 9.4.1**

Es un motor de base de datos surgido en 1986 con el lanzamiento de su primera versión. Este gestor es altamente potente y posee prestaciones y funcionalidades equivalentes a otros de carácter comercial. Es más completo que MySQL ya que permite métodos almacenados, restricciones de integridad y vistas. Es un sistema de gestión de base de datos relacional



orientada a objetos de *software* libre, publicado bajo la licencia BSD (*Berkeley Software Distribution*). PostgreSQL es un sistema de bases de datos objeto-relacional con características de los mejores sistemas de bases de datos comerciales, es libre y su código fuente completo está disponible. Algunas de sus características principales: alta concurrencia, amplia variedad de tipos nativos, uso de disparadores, funciones de Ventanas, expresiones de tablas comunes y consultas recursivas, instalaciones ilimitadas, estabilidad y confiabilidad, gran soporte a proveedores, extensible, multiplataforma, diseño para ambientes de amplio volumen (Gehrke, 2012).

Realizado el estudio de las ventajas del anterior gestor de bases de datos se hace recomendable el uso de PostgreSQL dada sus ventajas fundamentales de uso libre y soportar varias plataformas, lo que aumenta su potencia de uso y aplicación en distintos sistemas operativos.

#### **1.4 Conclusiones Parciales**

El presente capítulo ha posibilitado introducir y conocer aspectos fundamentales que nos permite adentrarnos en el proceso de cálculos de criticidad en el mundo del mantenimiento, así como a las principales funcionalidades que deben de ser creadas para la informatización del proceso. Permitted acercarse a los principales conceptos asociados al entorno y objetivos de trabajo, así como al alcance del mismo. Se valoran las principales herramientas a utilizar, así como las tendencias actuales en el campo de estudio tanto nacional como internacional. Se establecieron comparaciones con elementos similares para corroborar la eficacia de la selección propuesta y se validaron los posibles resultados finales a obtener. La utilización de PostgreSQL en su versión 9.4.1 garantiza una adecuada gestión de configuración y almacenamiento de la información. El uso de Visual Paradigm for UML 8.0 como herramienta CASE y la metodología de desarrollo para la actividad productiva de la UCI garantizan diseñar y elaborar la documentación necesaria para guiar el desarrollo del sistema de forma ágil y metódica.



## **CAPÍTULO II: ANÁLISIS Y DISEÑO DE LA PROPUESTA DE SOLUCIÓN**

En este capítulo se realiza una descripción general de la propuesta de solución a través de los requisitos funcionales y no funcionales, así como las particularidades del diseño. Se muestran aspectos esenciales de la arquitectura del sistema y los patrones de diseño empleados. También se hace alusión a los artefactos de la fase de Ejecución (requisitos, diseño).

### **2.1 Descripción de la propuesta de solución**

El sistema de cálculo de criticidades está enfocado en permitirle al ingeniero en mantenimiento o mantenedor desempeñar sus labores de una manera más sencilla, rápida y eficiente, para poder realizar estos cálculos se emplean varias fórmulas parametrizadas ya validadas. El sistema contará con una interfaz sencilla lo cual permitirá una fácil interacción, estará dividido por módulos, restringiendo el uso de un único módulo de acuerdo al contexto operacional de cada empresa en específico, fortaleciendo así su integración y seguridad. Los expertos en mantenimiento tendrán la opción de importar desde una hoja de cálculo los activos a los cuales le desea realizar el análisis de criticidad, o podrán ser capaces de insertarlos uno a uno manualmente, además poseerán la capacidad de modificar sus atributos una vez estén en el sistema, podrán eliminarlos, y realizar búsquedas parametrizadas.

Una vez que el mantenedor haya realizado uno o varios cálculos, sea cual sea el módulo en que se encuentre, éste tendrá la capacidad de graficar dichos valores de criticidad, exportarlos en distintos formatos y realizar comparaciones entre ellos. Además, el sistema almacenará un registro de todos los análisis de criticidad realizados, de manera tal que el mantenedor pueda consultar valores de criticidad con la fecha en que se obtuvieron, así como los parámetros que fueron seleccionados para realizar dicho análisis. También tendrá la opción de generar reportes y exportarlos en formato PDF con todos estos parámetros y valores. El mantenedor podrá además resetear los valores de criticidad de todos los activos que se encuentren en la base datos.





## 2.2 Captura de requisitos

La ingeniería de requisitos es el conjunto de actividades implicadas en descubrir, documentar y mantener un conjunto de requisitos del *software* a desarrollar. La captura de los mismos es un proceso en el cual los datos son extraídos a partir de la aplicación de técnicas de captura de información (Martínez, 2012).

En la fase de Ejecución de la metodología seleccionada para el desarrollo de la solución se define la disciplina *Requisitos*, esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. La entrevista, tormenta de ideas y estudio bibliográfico o documental fueron las técnicas empleadas para la captura de requisitos. La entrevista permitió apoderarse del conocimiento necesario para un completo dominio del problema, entendimiento del negocio y comprender los objetivos de la solución. El uso de la técnica tormenta de ideas, permitió acumular nuevas ideas y/o información, proporcionadas por varios expertos. Mientras que el estudio documental realizado, a través de consultas bibliográficas de varias tesis relacionadas con el tema en cuestión, posibilitó definir nuevas funcionalidades y estudiar los algoritmos matemáticos que se proponían para realizar análisis de criticidad.

### 2.2.1 Requisitos funcionales

Un requisito funcional es una condición o capacidad que debe ser alcanzada o poseída por un sistema o componente de un sistema, para satisfacer un contrato, estándar u otro documento impuesto formalmente (Jacobson, 2000). A partir del estudio y las investigaciones realizadas a la herramienta anteriormente desarrollada en la UCI sobre el marco de trabajo Sauxe, la cual englobaba los procesos de análisis de criticidad de activos en plantas de productos biológicos, grupos electrógenos, equipos especiales de aeropuertos, instalaciones hoteleras y plantas de coque, se reutilizaron los requisitos funcionales previamente definidos en esta, se modificaron algunos y se agregaron otros en consideración conjunta del equipo de desarrollo y el cliente. Además, se realizó también un estudio de los procesos de análisis de criticidad de activos en plantas termoeléctricas. A continuación, se muestra una lista con los requisitos funcionales del sistema:

*Tabla 1: Listado de Requisitos Funcionales.*

No.	Requisito	Descripción
<b>RF#1</b>	Importar activos	Permite importar cualquier cantidad de activos desde una hoja de cálculos.
<b>RF#2</b>	Listar activos	Permite listar todos los activos importados o insertados previamente.
<b>RF#3</b>	Buscar activos	Permite realizar una búsqueda de un activo específico dentro de toda la lista de activos, tanto por su nombre, como por su código, área o valor de criticidad.
<b>RF#4</b>	Ordenar activos	Permite ordenar la lista de los activos insertados o importados previamente teniendo en cuenta el código, el nombre, el área, los valores de criticidad y/o complejidad.
<b>RF#5</b>	Eliminar valores de criticidad	Permite eliminar de forma permanente del sistema todos los valores de criticidad de los activos.
<b>RF#6</b>	Eliminar valores de complejidad	Permite eliminar de forma permanente del sistema todos los valores de complejidad de los activos.
<b>RF#7</b>	Adicionar activo	Permite adicionar un activo.
<b>RF#8</b>	Modificar activo	Permite modificar sólo los parámetros iniciales de los activos importados o adicionados manualmente (nombre, localización o área, código).
<b>RF#9</b>	Eliminar activo	Permite eliminar de forma permanente un activo específico previamente seleccionado.
<b>RF#10</b>	Calcular criticidad	Permite realizar el cálculo de criticidad de un activo específico previamente seleccionado por el mantenedor, luego de haber elegido los parámetros deseados.
<b>RF#11</b>	Calcular complejidad	Permite realizar el cálculo de complejidad de un activo específico previamente seleccionado.
<b>RF#12</b>	Modificar criticidad	Permite actualizar el valor de criticidad calculado.
<b>RF#13</b>	Modificar complejidad	Permite actualizar el valor de complejidad calculado.
<b>RF#14</b>	Mostrar historial de criticidad por activo	Permite mostrar un historial con todos los valores de criticidad previamente calculados a un activo en específico, así como la



		fecha en que se realizaron dichos cálculos y los parámetros que fueron seleccionados para llevar a cabo el mismo.
<b>RF#15</b>	Mostrar historial de complejidad por activo	Permite mostrar un historial con todos los valores de complejidad previamente calculados a un activo en específico, así como la fecha en que se realizaron dichos cálculos y los parámetros que fueron seleccionados para llevar a cabo el mismo.
<b>RF#16</b>	Mostrar parámetros del historial por activo	Permite mostrar los parámetros que fueron seleccionados por el mantenedor para llegar a obtener el valor de criticidad o complejidad previamente calculado.
<b>RF#17</b>	Mostrar reporte de criticidad	Permite generar un reporte con todos los valores de criticidad de todos los activos.
<b>RF#18</b>	Mostrar reporte de complejidad	Permite generar un reporte con todos los valores de complejidad de todos los activos.
<b>RF#19</b>	Exportar reporte de criticidad	Permite exportar un reporte con todos los valores de complejidad obtenidos de todos los activos insertados en el sistema, en formato PDF.
<b>RF#20</b>	Exportar reporte de complejidad	Permite exportar un reporte de complejidad en formato PDF.
<b>RF#21</b>	Graficar análisis de criticidad	Permite graficar el último valor de criticidad de todos los activos a los cuales se les haya calculado la criticidad previamente.
<b>RF#22</b>	Graficar análisis de complejidad	Permite graficar el último valor de complejidad de todos los activos a los cuales se les haya calculado la criticidad previamente.
<b>RF#23</b>	Graficar historial de criticidad	Permite graficar todos los valores de criticidad previamente calculados a un activo en específico.
<b>RF#24</b>	Graficar historial de complejidad	Permite graficar todos los valores de complejidad previamente calculados a un activo en específico.
<b>RF#25</b>	Graficar análisis de criticidad contra complejidad	Permite graficar todos los valores de criticidad contra todos los valores de complejidad previamente calculados a un activo.
<b>RF#26</b>	Exportar gráfica de	Permite exportar la gráfica de los valores de criticidad de



	criticidad	todos los activos a los cuales se les haya calculado la criticidad previamente, en distintos formatos como por ejemplo: PDF, JPEG, PNG, BMP, SVG.
<b>RF#27</b>	Exportar gráfica de complejidad	Permite exportar la gráfica de los valores de complejidad de todos los activos a los cuales se les haya calculado la complejidad previamente, en distintos formatos como por ejemplo: PDF, JPEG, PNG, BMP, SVG.
<b>RF#28</b>	Exportar gráfica de criticidad contra complejidad	Permite exportar la gráfica de los valores de criticidad contra los valores de complejidad de todos los activos a los cuales se les haya calculado la criticidad y complejidad previamente, en distintos formatos como por ejemplo: PDF, JPEG, PNG, BMP, SVG.
<b>RF#29</b>	Filtrar reportes por áreas	Permite filtrar por áreas el reporte de criticidad y/o complejidad a generar, de manera tal que el mantenedor solo obtendrá en formato PDF los datos de los activos que pertenezcan a las áreas seleccionadas, por defecto, el sistema filtra todas las áreas.
<b>RF#30</b>	Mostrar/Ocultar cuadrantes	Permite mostrar u ocultar los activos que se encuentren en cualquiera de los cuadrantes que se muestran en la gráfica.
<b>RF#31</b>	Exportar reporte del historial	Permite exportar un reporte generado tanto de criticidad como de complejidad en consecuencia con el historial, en formato PDF.
<b>RF#32</b>	Imprimir gráficas	Permite el envío para impresión de las gráficas generadas hacia la impresora instalada en la computadora que se esté accediendo a la aplicación.



### 2.2.2 Requisitos no funcionales

Los requisitos no funcionales restringen el sistema en desarrollo y el proceso de desarrollo de *software* que se debe utilizar. Pueden ser requisitos del producto, organizacionales o externos. A menudo están relacionados con las propiedades emergentes del sistema y, por lo tanto, se aplican al sistema (Sommerville, 2005).

Son capacidades o cualidades que el producto debe tener, o sea, características que hagan al producto atractivo, rápido, usable o confiable. Están estrechamente vinculados a los requisitos funcionales, puesto que una vez que está definido lo que el sistema debe hacer, es necesario especificar como ha de hacerlo. Pueden llegar a marcar la diferencia entre un producto bien aceptado por los clientes y usuarios o uno de poca o ninguna calidad y aceptación (Jacobson, 2000).

Después del estudio realizado para el desarrollo del módulo cálculo de criticidad de activos en plantas de productos biológicos, grupos electrógenos, equipos especiales de aeropuertos, instalaciones hoteleras, plantas de coque y plantas termoeléctricas, se identificaron un conjunto de requisitos no funcionales de la aplicación a implementar, los cuales fueron validados por el cliente, ya que se definieron en conjunto con el mismo.

#### ✓ **Apariencia o interfaz externa**

**RNF 1:** Debe poseer un diseño adaptativo, es decir debe adaptarse a las dimensiones de la pantalla.

**RNF 2:** La interfaz debe tener un diseño uniforme para todos los módulos.

**RNF 3:** Los mensajes, títulos y demás textos que aparezcan en la interfaz del sistema deben aparecer en idioma español.

**RNF 4:** La entrada incorrecta de datos será señalada al usuario claramente, indicándole los campos donde se encuentra el error.

#### ✓ **Usabilidad**

**RNF 5:** Utilizar campos de selección en la interfaz en los casos que sea posible.

**RNF 6:** El sistema deberá desarrollarse sobre una plataforma web, la cual permitirá al usuario tener acceso a la aplicación desde cualquier navegador.

#### ✓ **Software**

**RNF 7:** Por parte del cliente; el navegador web que se utilice para interactuar con la aplicación deberá tener soporte para HTML5. Se recomienda utilizar Mozilla Firefox en su versión 30 o superior.



**RNF 8:** Por parte del servidor; como herramienta para la gestión, procesamiento, almacenamiento y consulta de la base de datos se deberá instalar como Gestor de Base de Datos, PostgreSQL. Como sistema operativo se deberá tener instalado Ubuntu 14.4 o una distribución superior.

✓ **Hardware**

**RNF 9:** Por parte del cliente, como requisitos mínimos; un procesador Intel Pentium Dual Core a 1.6 GHz y 512MB de memoria RAM.

**RNF 10:** Por parte del servidor, como requisitos mínimos, un procesador Intel Pentium Dual Core a 2.5 GHz, capacidad para el disco duro de 5GB y 2GB de Memoria RAM.

✓ **Disponibilidad**

**RNF 11:** El sistema debe estar disponible las 24 horas del día para trabajar en el momento deseado, aunque pudiera ser a consideración del cliente. Para ello se recomienda que la PC que se seleccione como servidora sea de última generación y posea buenas prestaciones; que se garantice un pleno funcionamiento de las conexiones de red; y que se cuente con un mecanismo de respaldo en caso de ausencia del fluido eléctrico.

✓ **Confiabilidad**

**RNF12:** Los cálculos para la implementación de los modelos matemáticos deben estar en correspondencia con los estudiados, con el objetivo de garantizar que los resultados que se obtengan sean los esperados.

✓ **Diseño e Implementación**

**RNF13:** El sistema debe poseer una arquitectura cliente-servidor.

**RNF14:** El sistema debe ser implementado haciendo uso del lenguaje de programación Python sobre el IDE de desarrollo PyCharm.

**RNF 15:** Durante el proceso de implementación de la herramienta se deberá emplear el estándar de codificación CamelCase.

✓ **Soporte**

**RNF 16:** El sistema deberá ser escalable ya que se podrán añadir funcionalidades sin que se vean afectadas las ya implementadas.

**RNF 17:** La herramienta recibirá mantenimiento en el período de tiempo determinado por el equipo de trabajo y los clientes involucrados en su desarrollo.

**RNF 18:** La herramienta debe ser de fácil instalación.

✓ **Rendimiento**

**RNF 19:** La aplicación debe tener una respuesta rápida ante cualquier solicitud del usuario, máximo 5 segundos en el caso de generar un reporte por todas las áreas habiendo una cantidad



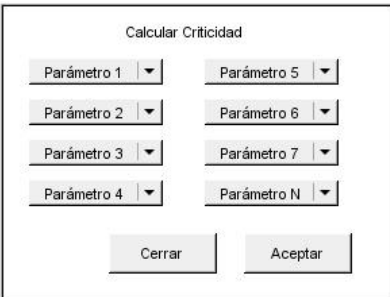
sustancial de activos en el sistema (superior a 10000), considerándose este como el peor de los casos.



### 2.2.3 Historias de usuarios.

Las Historias de Usuarios (HU) según la metodología de *software* empleada en la presente solución se define como una de las formas que se utilizan para encapsular requisitos, esta forma se evidencia en el cuarto escenario condicionado por *Modelado del Negocio*. En el desarrollo de la herramienta propuesta se incluyeron nuevos requisitos como, por ejemplo: ordenar activos, buscar activos, eliminar valores de criticidad y complejidad, mostrar historial de criticidad y complejidad por activo, mostrar parámetros del historial por activo, exportar gráfica de criticidad y complejidad, filtrar reportes por áreas, mostrar/ocultar cuadrantes de las gráficas, exportar reporte del historial e imprimir gráficas. Los demás requisitos se mantienen igualmente como los descritos en la versión anteriormente implementada de la herramienta. A continuación, se muestra la Historia de Usuario del requisito funcional “*Calcular Criticidad*” proponiéndose como ejemplo de dicho requisito dentro del módulo Plantas Termoeléctricas, ya que se considera sea la funcionalidad más significativa, debido a que es la principal tarea para satisfacer el propósito para el cual fue desarrollada la herramienta. Las restantes HU se encuentran en la sección Anexos.

*Tabla 2: Historia de Usuario del requisito funcional calcular criticidad.*

Historia de Usuario	
<b>Número:</b> HU_10	<b>Nombre del requisito:</b> Calcular Criticidad
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 20horas
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 13 horas
<b>Descripción:</b> Permitirá realizar el cálculo de criticidad de un activo específico previamente seleccionado por el mantenedor, luego de haber elegido los parámetros deseados.	
<b>Observaciones:</b> N/A	
<b>Prototipo de interfaz:</b> 	





#### 2.2.4 Validación de requisitos

Con el objetivo de verificar si los requisitos del *software* obtenidos definen el sistema que el cliente desea, se llevó a cabo un proceso de validación de los mismos, para el cual se emplearon las siguientes técnicas:

- ✓ **Revisiones técnicas formales:** se realizaron revisiones técnicas formales que contribuyeron al monitoreo del avance en el desarrollo de la herramienta, donde se iba realizando un chequeo por encuentro a cada uno de los requisitos funcionales implementados por parte del equipo de desarrollo para comprobar que se correspondían con los descritos por el cliente y si satisfacía sus necesidades. Estas revisiones internas generaron algunas no conformidades de tipo técnicas y de ortografía, las cuales fueron corregidas satisfactoriamente en el tiempo reglamentado, finalmente el cliente quedó satisfecho con el trabajo efectuado y se aprobaron los requisitos, generándose el acta de aceptación, la cual se encuentra plasmada en la sección Anexos.
- ✓ **Construcción de prototipos:** mediante prototipos teóricos, luego prototipos no funcionales y por último prototipos funcionales se le mostró al cliente varios modelos del sistema a través de 3 iteraciones. En la primera iteración se detectaron errores de conceptos relacionados con el negocio a implementar, en la segunda iteración se señalaron imperfecciones de diseño y maquetación, y en la tercera iteración se detectaron errores de funcionalidad. Todos los errores y señalamientos encontrados fueron arreglados inicialmente en un 45%, dando lugar a que finalmente se corrigieran en un 100%. Todo lo anteriormente expuesto le permitió al cliente tener una visión preliminar de cómo sería la herramienta web, que luego se fortaleció con una versión funcional del sistema y a través de la interacción con ésta se comprobó si placía sus necesidades, finalmente los prototipos fueron aprobados por el cliente con un alto nivel de satisfacción.

#### 2.3 Arquitectura del sistema

La arquitectura de *software* demuestra la organización, funcionamiento y conexión entre las partes de un sistema. Su objetivo principal es garantizar un mejor desempeño en el desarrollo de las aplicaciones. Proporciona robustez, portabilidad y flexibilidad a la aplicación. Es considerado el elemento de enlace entre los requerimientos y la implementación del sistema (Gamma, 2009).



### 2.3.1 Patrón de Arquitectura

Los patrones de arquitectura están orientados a representar los diferentes elementos que componen una solución de *software* y las relaciones entre ellos, forman parte de la llamada Arquitectura de Software (arquitectura lógica de un sistema), que comprende: el diseño de más alto nivel de la estructura del sistema, los patrones y abstracciones necesarios para guiar la construcción del *software* de un sistema; los fundamentos para que analistas, diseñadores, programadores, beta testers, trabajen en una línea común que permita cubrir restricciones y alcanzar los objetivos del sistema.

La arquitectura de *software* define, de manera abstracta, los componentes que llevan a cabo alguna tarea de proceso de información, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué ordenador tendrá asignada cada tarea (Wilson, 2011). Existen al menos tres vistas absolutamente fundamentales en cualquier arquitectura: la estática, que describe qué componentes tiene la arquitectura; la funcional, que describe qué hace cada componente; y la dinámica, que describe cómo se comportan los componentes a lo largo del tiempo y cómo interactúan entre sí. El patrón de arquitectura debe contener de alguna manera, información para que estas tres vistas fundamentales estén presentes y puedan servir para la implementación de soluciones basadas en él.

#### Modelo-Plantilla-Vista (MTV)

Django es conocido como un Framework MTV, del inglés "*Model-Template-View*" (Modelo Plantilla Vista), que no es más que una variación del patrón Modelo-Vista-Controlador, e intenta ser lo más funcional posible. En comparación con el patrón MVC, el modelo en Django sigue siendo modelo; a la vista se le denomina plantilla y en el caso del controlador se le denomina vista (Nieto, 2015).

En el patrón de arquitectura MTV:

- ✓ **M** significa "*Model*" (**Modelo**), la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cuál es el comportamiento que tiene, y las relaciones entre los datos.

El modelo define los datos almacenados, se encuentra en forma de clase de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros; también posee métodos. Todo esto nos permite indicar y controlar el comportamiento de los datos.

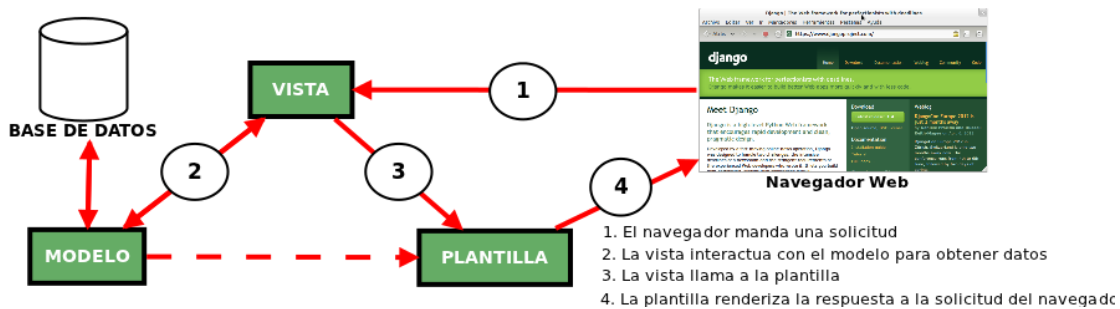


- ✓ **T** significa "Template" (**Plantilla**), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: como algunas cosas son mostradas sobre una página web u otro tipo de documento.

La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solo crea contenido en HTML sino también XML, CSS, JavaScript. La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web.

- ✓ **V** significa "View" (**Vista**), la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: puedes pensar en esto como un puente entre el modelo y las plantillas.

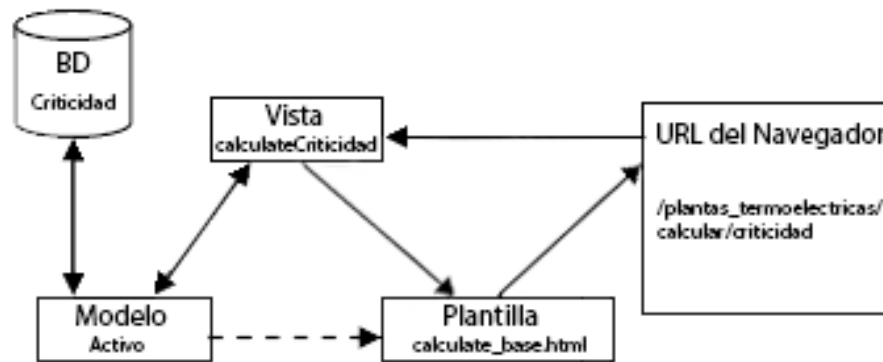
La vista se presenta en modo de funciones en Python y su propósito es determinar qué datos serán visualizados. El ORM de Django permite escribir código Python en lugar de SQL para realizar las consultas que necesita la vista. La autenticación de servicios y la validación de datos a través de formularios son también tareas de las que se encarga la vista (Montero, 2012).



**Figura 1:** Arquitectura Modelo-Plantilla-Vista (MTV).

Un ejemplo claro de cómo se observa esta arquitectura que se define en Django dentro del sistema de cálculo de criticidades lo constituye el siguiente ejemplo;

La *url*:/plantas\_termoelectricas/calcular/criticidad (que se comporta como el navegador web), realiza una petición a la *view* (vista) "calculateCriticidad", la cual es renderizada por el *template* (plantilla) "calculate\_base.html" y este le envía a la *url* como respuesta de la petición realizada, a su vez hace uso del *model* (modelo) "Activo" que se comunica con la tabla "Criticidad" de la base de datos para almacenar el cálculo realizado.



*Figura 2: Ejemplo Arquitectura Modelo-Plantilla-Vista (MTV) a través de la funcionalidad calcular criticidad en el módulo Plantas Termoeléctricas del sistema.*

## 2.4 Patrones de diseño

En términos generales, un patrón es un conjunto de "elementos" que se repiten de manera predecible. Ahora bien, los patrones de diseño son situaciones o problemas observados de manera recurrente, y asociados a ellas, una solución óptima. Constituyen un conjunto de información que proporciona respuesta a un grupo de problemas similares, es decir, un patrón es una solución a un problema dentro de un contexto determinado (Jiménez, 2011).

Además del uso del patrón arquitectónico MTV, en la solución propuesta se emplean distintos tipos de patrones de diseño como son: de asignación de responsabilidad y de comportamiento; los cuales se describen a continuación:

### 2.4.1 Patrones de Diseño de Asignación de Responsabilidades (GRASP)

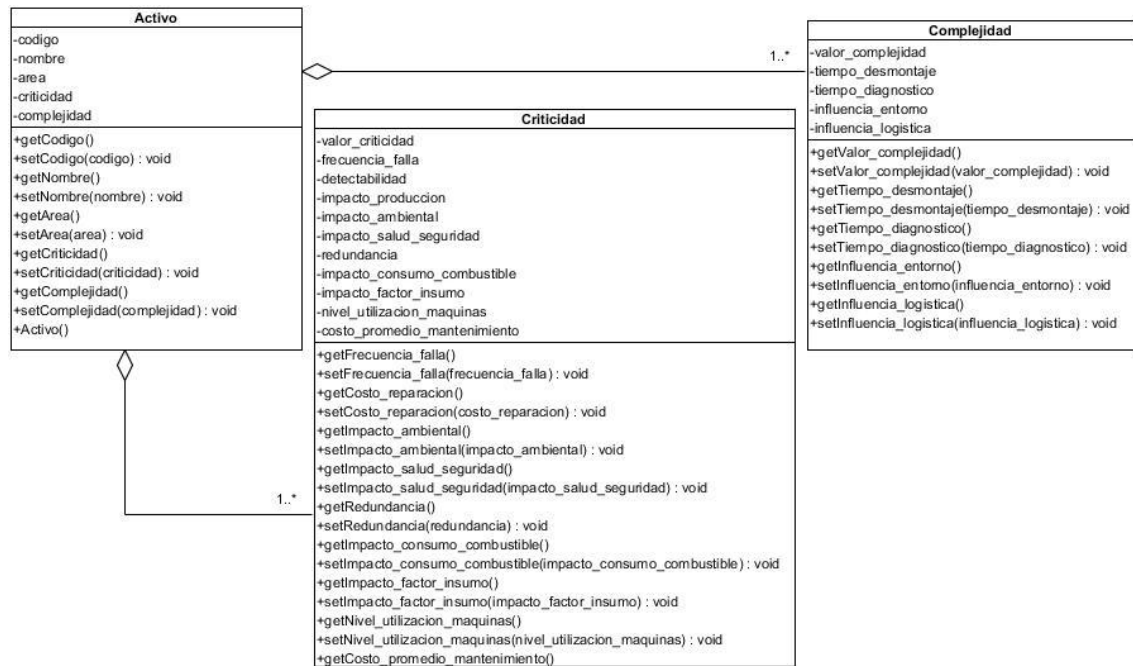
Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable (Grosso, 2011).

Los patrones de este tipo que se emplearon en la implementación de la herramienta se describen a continuación:

- ✓ **Experto en información:** Expresa la intuición común de que los objetos hacen las cosas relacionadas con la información que tienen. Se mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener. Se distribuye el comportamiento entre las clases que

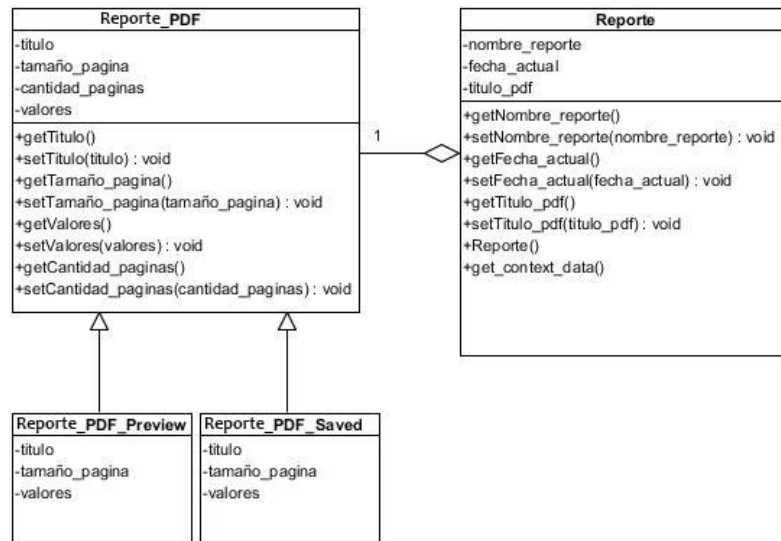


contienen la información requerida, por tanto, se estimula las definiciones de clases más cohesivas y ligeras que son más fáciles de entender y mantener (Grosso, 2011). Este patrón es aplicado en todas las clases debido a que cada una de ellas es experta pues contienen la información necesaria para cumplir con las responsabilidades que le fueron asignadas; facilitando así el entendimiento, extensión y mantenimiento del sistema. A continuación, se muestra un diagrama UML donde se evidencia su uso a través de la clase Activo que interactúa con las clases Criticidad y Complejidad.



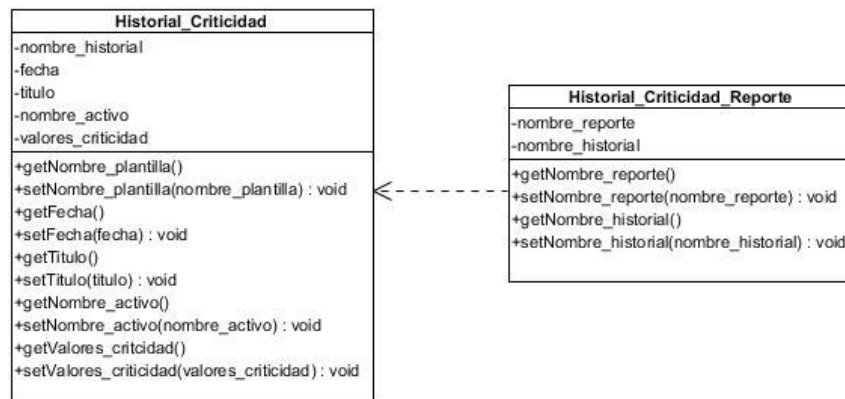
**Figura 3:** Ejemplo del patrón Experto en Información empleado en la implementación de la herramienta a través de un diagrama UML.

- ✓ **Creador:** Se asigna la responsabilidad a una clase de crear cuando contiene, agrega, compone, almacena o usa otra clase, esto brinda una alta posibilidad de reutilizar la clase creadora (Grosso, 2011). El patrón se evidencia en las clases controladoras que, para cada uno de los módulos o funcionalidades de la aplicación, son las encargadas de crear las instancias de los objetos que manejan, favoreciendo así la reutilización y el bajo acoplamiento. Un ejemplo de ello se evidencia en la generación de un report donde se instancia la clase PDF y se crean dos objetos de tipo pdf.



**Figura 4:** Ejemplo del patrón Creador empleado en la implementación de la herramienta a través de un diagrama UML.

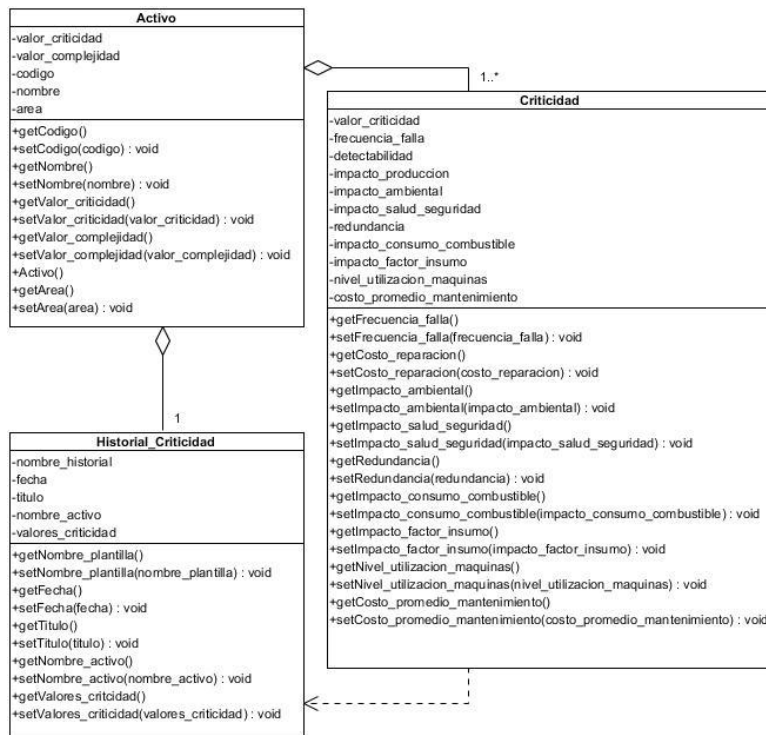
- ✓ **Bajo Acoplamiento:** Este patrón garantiza que las clases estén lo menos ligadas posible entre sí, de tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (Grosso, 2011). Mediante el uso de este patrón se proporciona un bajo acoplamiento en el diseño debido a que las clases existentes tienen asignadas responsabilidades de tal forma que estas no dependan en gran medida de otras, permitiendo de esta forma tener un sistema más robusto y de fácil mantenimiento. A continuación, se muestra un ejemplo donde se presenta una de las formas de acoplamiento, que es: Clase A es subclase de B, evidenciándose en clase *logsCriticidadReporte* la cual es una subclase de la clase *logsCriticidad*.



**Figura 5:** Ejemplo del patrón Bajo Acoplamiento empleado en la implementación de la herramienta a través de un diagrama UML.



- ✓ **Alta Cohesión:** Define que la información almacenada en una clase debe ser coherente y estar relacionada con esta. Propone que no se debe saturar una clase de métodos, sino asignar las responsabilidades a cada clase correspondiendo a la información que almacena (Grosso, 2011). Este patrón se utiliza debido a que a cada una de las clases se le asignaron responsabilidades de tal forma que estén estrechamente relacionadas entre sí y no realicen un trabajo excesivo. A continuación, se muestra un ejemplo donde se evidencia lo antes descrito.



*Figura 6: Ejemplo del patrón Alta Cohesión empleado en la implementación de la herramienta a través de un diagrama UML.*

### 2.4.2 Patrones de Comportamiento (GOF)

Los **patronesGoF** por sus siglas en inglés de *Gang of Four*, son en el campo del diseño orientado a objetos los más conocidos y usados en la actualidad. Estos describen las formas en las que pueden ser organizados los objetos para trabajar unos con otros (Larman, 2003). Para la implementación de la solución propuesta se utilizaron los patrones active record y decorator que se explican a continuación:

- ✓ **Active Record:** Es un patrón en el cual, el objeto contiene los datos que representan una colección, además de encapsular la lógica necesaria para acceder a la base de datos. De esta forma el acceso a datos se presenta de manera uniforme a través de la aplicación.





Una clase Active Record consiste en el conjunto de propiedades que representa las columnas de la tabla, más los típicos métodos de acceso como las operaciones CRUD (en inglés: Create, Read, Update and Delete), búsqueda, validaciones, y métodos de negocio(Fowler, 2003). Un ejemplo de este patrón en la propuesta de solución se muestra a continuación:



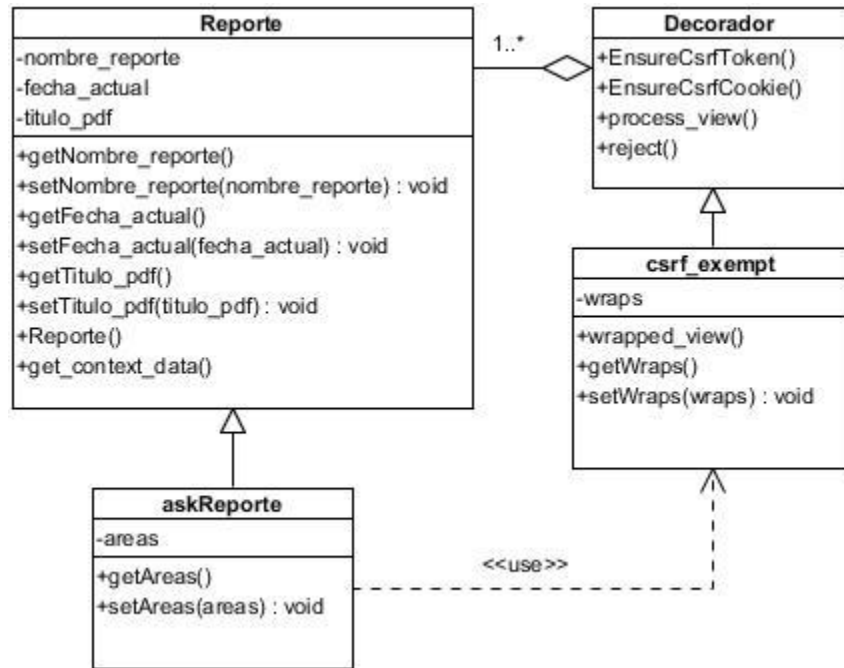
**Figura 7:** Ejemplo del patrón Active Record empleado en la implementación de la herramienta a través de un diagrama UML.

- ✓ **Decorator:** Es un patrón de tipo estructural encargado de asociar responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades. Brinda mayor flexibilidad que la herencia estática, permitiendo, entre otras cosas, añadir una funcionalidad dos o más veces. Propicia concentrar en lo alto de la jerarquía de clases guiadas por las responsabilidades. De esta forma las nuevas funcionalidades se componen de piezas simples que se crean y se combinan con facilidad, independientemente de los objetos cuyo comportamiento extienden(Fowler,





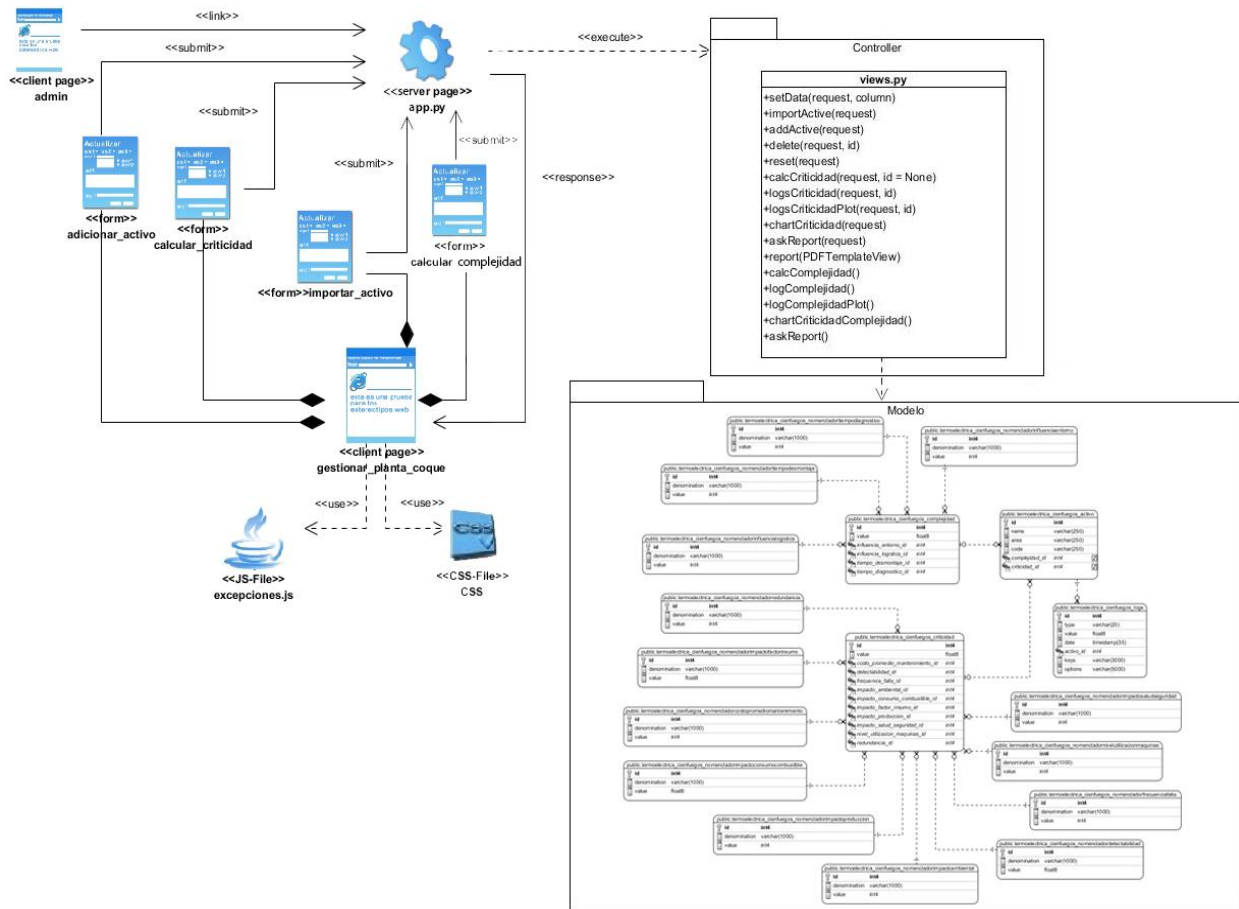
2003). A continuación, se muestra un ejemplo de cómo se evidencia este patrón en el sistema propuesto a solución:



**Figura 8:** Ejemplo del patrón Decorator empleado en la implementación de la herramienta a través de un diagrama UML.

## 2.5 Diagrama de clases del diseño con estereotipos web.

El propósito del diagrama es el de representar las diferentes clases que se utilizan en el sistema, así como las relaciones que existen entre estas. Para el sistema, se elaboraron un total de 6 diagramas correspondientes a cada uno de los módulos implementados, a continuación, se muestra el diagrama de clases del diseño con estereotipos web referente al módulo Plantas Termoeléctricas. Los demás diagramas se muestran en la sección Anexos.



**Figura 9:** Diagramas de clases del diseño con estereotipos web: Gestionar Plantas Termoeléctricas.

A partir de la representación gráfica de este módulo a través del diagrama de clases con estereotipos web se evidencia el uso del patrón de arquitectura Modelo-Plantilla-Vista propuesto por el marco de trabajo empleado en el desarrollo de la propuesta de solución. En el mismo, los formularios, la página cliente y la página servidora contenida implícitamente dentro de las Plantillas se comunican con las Vistas, las cuales actúan como controladoras dentro de la herramienta, que es donde se encuentra implementado todos los métodos y funcionalidades de la lógica de negocio. Las Vistas se comunican con el Modelo, donde se encuentra toda la estructura de datos, responsable del manejo de la información.



## 2.6 Modelo de base de datos

El modelo de datos es un conjunto de conceptos que sirven para describir la estructura, semántica y las relaciones existen entre las entidades de una base de datos importantes para el funcionamiento del negocio y muestra los datos que serán contenidos en el sistema (Fernández, 2011).

A continuación, se muestra el modelo de base de datos del módulo Plantas Termoeléctricas, la cual permite realizar la representación lógica de los datos procesados por el mismo. Este modelo muestra las entidades de datos por las que está formado el mismo, los atributos asociados a cada una de ellas y las relaciones entre ellas. Los restantes modelos se encuentran plasmados en la sección Anexos.

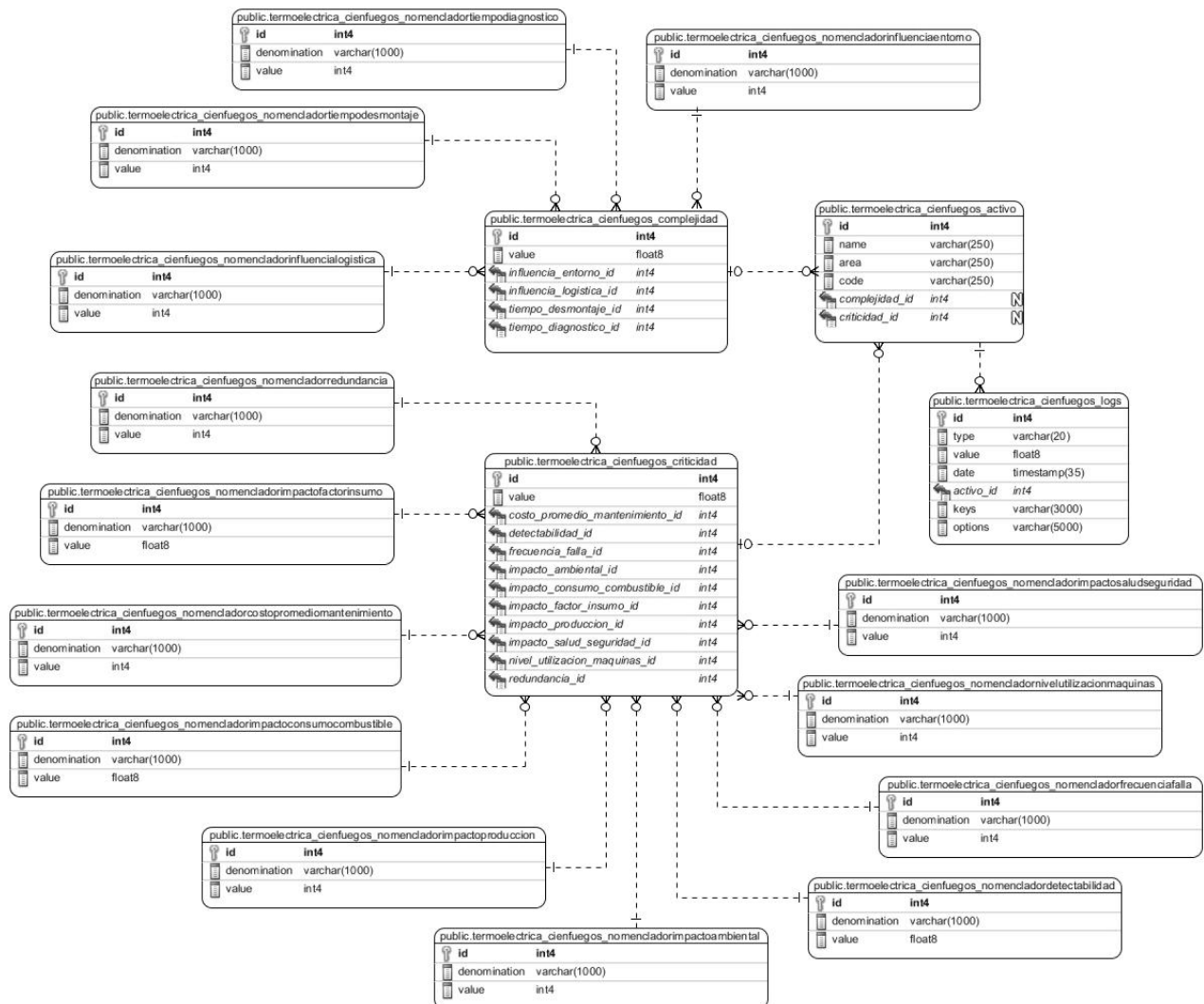


Figura 10: Modelo de Base de Datos del Módulo Plantas Termoeléctricas.



EL modelo está compuesto por cuatro tablas fundamentales que contienen la información del módulo anteriormente mencionado, las mismas son: activo, criticidad, complejidad e historial, las cuales se vinculan unas con otras a través de las relaciones uno a uno en el caso de las tablas activo e historial ya que un activo solo puede tener solo un historial. Las tablas activo, criticidad y complejidad se corresponden a través de la relación muchos a muchos, ya que un activo puede poseer varios valores de criticidad y complejidad los cuales son almacenados en el historial con la fecha en que se obtuvieron y los parámetros seleccionados para el cálculo. Las restantes tablas del modelo se corresponden a los nomencladores establecidos para la base de datos, es decir, a cada uno de los juegos de datos que el usuario o mantenedor debe seleccionar a la hora de realizar un análisis de criticidad, estos se atañen a través de la relación uno a muchos con las tablas criticidad y complejidad, puesto que para poder calcularle la criticidad a un activo existen varias opciones de selección dentro de los juegos de datos, pero se debe seleccionar uno de ellos.

### 2.6.1 Patrones de diseño de la base de datos.

- ✓ **Llaves subrogadas:** El uso de este patrón es bastante generalizado en la base de datos pues la mayoría de las entidades contienen una llave única entera auto-incremental, lo que facilita reducir el costo de las búsquedas en la base de datos. El uso de este patrón constituye una protección ante los cambios debido a que la lógica del negocio no está en las llaves y evita la contención (bloqueo) de la base de datos, debido a la rapidez de los mecanismos de generación que provee el sistema.

public.termoelectrica_cienfuegos_criticidad	
id	int4
value	float8
costo_promedio_mantenimiento_id	int4
detectabilidad_id	int4
frecuencia_falla_id	int4
impacto_ambiental_id	int4
impacto_consumo_combustible_id	int4
impacto_factor_insumo_id	int4
impacto_produccion_id	int4
impacto_salud_seguridad_id	int4
nivel_utilizacion_maquinas_id	int4
redundancia_id	int4

*Figura 11: Uso del patrón Llaves Subrogadas en la entidad termoelectrica\_cienfuegos\_criticidad.*



### **Conclusiones parciales**

En el presente capítulo se realizó la descripción de la aplicación mediante el uso de la metodología de desarrollo AUP-UCI en su segunda fase (Ejecución). Se presentó la implementación de una herramienta web para realizar cálculos de criticidad basada en los modelos de criticidad personalizados, teniendo en cuenta el problema y el estado del arte establecido en el Capítulo 1. Se realizó la captura de requisitos, aplicándose como técnicas de ingeniería de requisitos la entrevista, la tormenta de ideas y el estudio bibliográfico, las cuales permitieron definir un total de 32 RF, y 19 RNF. También se llevó a cabo el encapsulamiento y descripción de los requisitos funcionales a través del cuarto escenario de la metodología de desarrollo AUP-UCI mediante las Historias de Usuario (HU) definidas como punto de partida para el diseño y realización de los prototipos del sistema. Lo que se concretó con la elaboración de las Plantillas de Historias de Usuario, que aglutinan su descripción, junto al prototipo y el requisito al que responde cada HU.

Posteriormente se realizó un análisis sobre la arquitectura a la cual responde el sistema, resaltando el uso del patrón arquitectónico Modelo-Plantilla-Vista y el comportamiento de los componentes que interactúan de forma transversales a las capas para garantizar seguridad, tratamiento de excepciones, entre otros aspectos. Luego, se realizó un estudio sobre el uso de los patrones de diseño utilizados durante el desarrollo de la aplicación, destacando el empleo de los patrones alta cohesión, bajo acoplamiento, creador y active record, mediante los cuales se logró disminuir la complejidad en el proceso de implementación. Por último, para describir la estructura, semántica y las relaciones existen entre las entidades de la base de datos se elaboró el Modelo de Datos de la Aplicación y se validaron los requisitos, aplicando técnicas como construcción de prototipos y revisiones técnicas formales.



## **CAPÍTULO III: IMPLEMENTACIÓN Y VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN**

En este capítulo se describe el proceso de implementación de la aplicación en términos de componentes; donde se detalla mediante el diagrama de despliegue como quedará distribuido el sistema. Se muestran los resultados obtenidos de la aplicación después de aplicarle un conjunto de métricas y técnicas empleadas para validar los requisitos y el diseño de la propuesta de solución; además de los resultados alcanzados luego de la realización de varias pruebas. De ello, se generan los principales productos de trabajo definidos por la metodología, correspondientes a la fase de ejecución, en sus disciplinas (implementación y pruebas internas, de liberación y de aceptación).

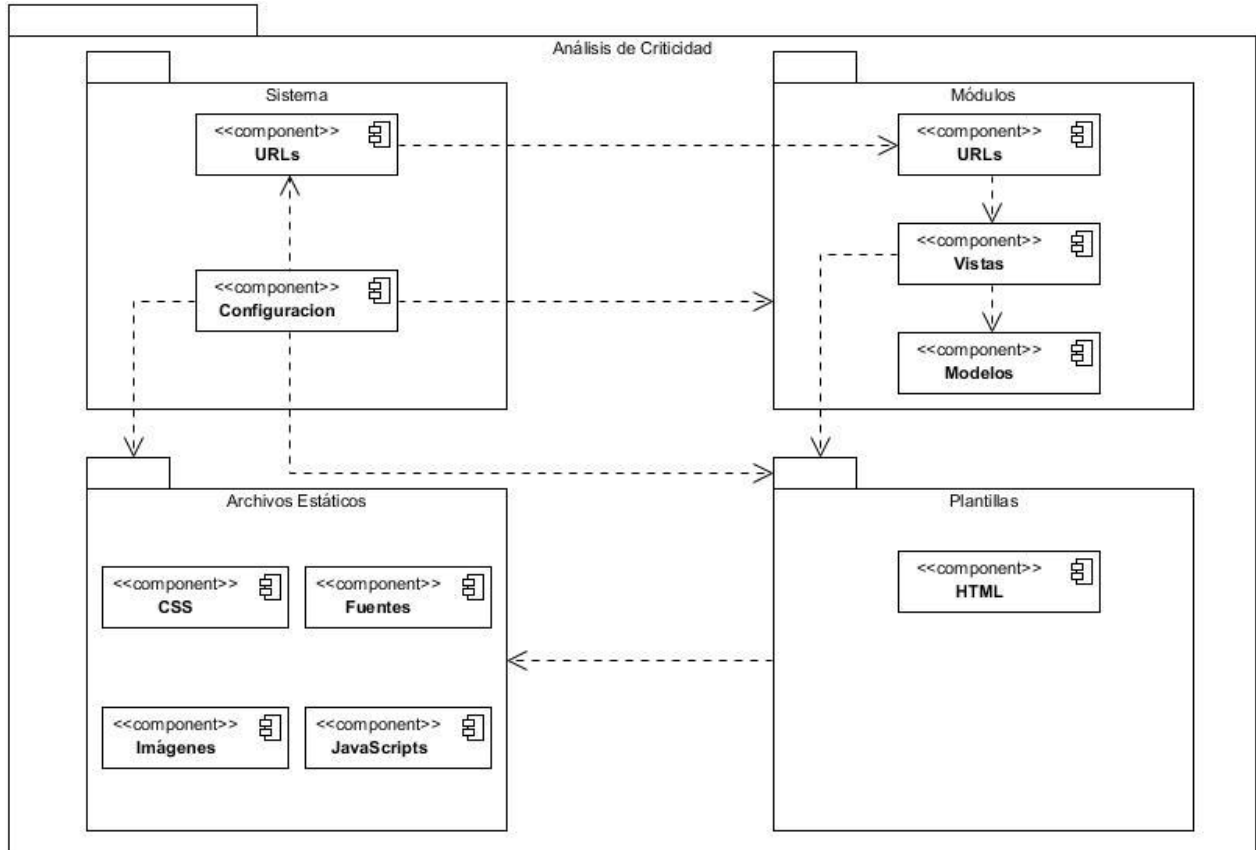
### **3.1 Implementación**

La implementación constituye una de las fases más importantes del desarrollo de *software*. En ella se toman como punto de partida los resultados obtenidos en el diseño, implementándose el sistema en términos de componentes como ficheros de código binario, código fuente, scripts y ejecutables. Su importancia reside en que se obtiene como consecuencia una herramienta informática o un sistema ejecutable, siendo esto uno de los principales objetivos en el desarrollo de *software* (Arizaca, 2011).

#### **3.1.1 Diagrama de componentes**

Un diagrama de componentes representa la separación de un sistema de *software* en componentes físicos (por ejemplo, archivos, módulos, paquetes, base de datos, código fuente, binario o ejecutable, y muestra las dependencias y organización existente entre estos componentes (Arizaca, 2011).

En el diagrama de componentes de la Figura 6, se evidencian las interacciones entre los componentes del sistema "Análisis de Criticidad", está compuesto por cuatro paquetes fundamentales; Sistema, Módulos, Plantillas y Archivos Estáticos. Estos paquetes con sus correspondientes componentes permiten un adecuado manejo de la seguridad, las configuraciones, las excepciones y errores del subsistema, así como de la información que se maneja.



**Figura 12:** Diagrama de Componentes.

El paquete Sistema maneja todas las configuraciones de la herramienta informática y contiene los componentes Configuración; encargado de registrar todas las configuraciones de los demás paquetes, y URLs; que contiene y manipula en su totalidad las URLs de toda la solución informática. El paquete Módulos compuesto por los componentes Vistas, Modelos y URLs, representa físicamente a cada uno de los 6 módulos por los que está compuesta la propuesta de solución, donde el componente URLs realiza peticiones a las vistas (donde se encuentra toda la lógica del negocio) y estas se comunican con los modelos y a su vez renderizan las peticiones al paquete *Templates*, el cual está compuesto por el componente HTML, que es donde se almacena todos el código fuente del sistema, y este paquete *Templates* hace uso del paquete Archivos Estáticos, donde se encuentran localizados todos los archivos *JavaScript*, los estilos CSS, las fuentes y las imágenes de la herramienta informática.

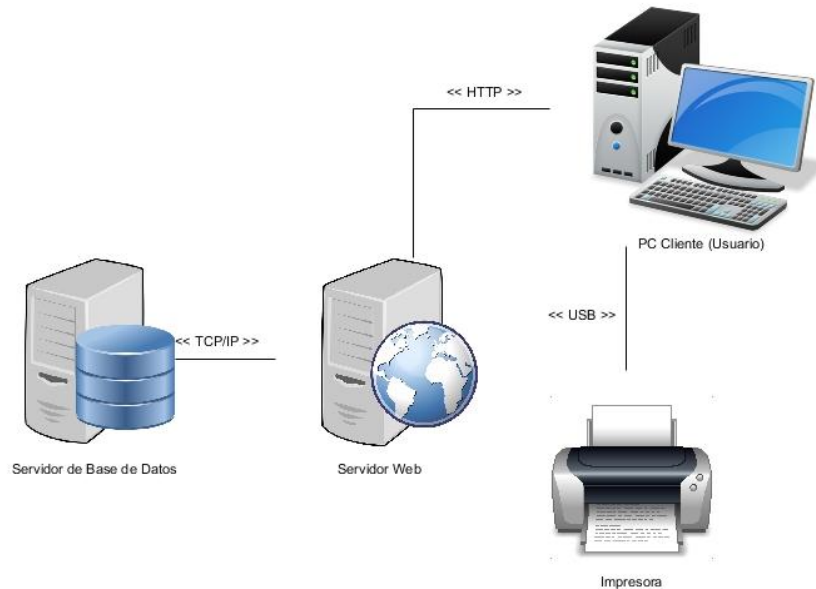
De igual forma, estos componentes pueden ser agrupados y organizados atendiendo a la arquitectura del sistema para lograr un mejor entendimiento de sus funciones, distribución e interacciones entre sí.



### 3.1.2 Diagrama de despliegue

Los diagramas de despliegue proveen una vista de cómo los componentes se despliegan en el transcurso de la infraestructura del sistema. Permite comprender la correspondencia entre la arquitectura de *software* y la de *hardware*. El siguiente diagrama de despliegue muestra la forma en que se comunicarán los componentes del sistema.

La PC cliente (usuario) se va a comunicar por el protocolo HTTP<sup>1</sup> para utilizar las funcionalidades que se encuentran en el Servidor Web, la cual cuenta con una impresora con conexión USB donde se podrá obtener de forma física todos los cálculos realizados e información generada por la aplicación. Este Servidor Web establecerá comunicación con el Servidor de Base de Datos por medio del protocolo TCP/IP<sup>2</sup> para la obtención de la información que maneja el sistema,



**Figura 13:** Diagrama de Despliegue.

<sup>1</sup> HTTP: Protocolo de transferencia de hipertexto.

<sup>2</sup> TCP/IP: Protocolo de Control de Transmisión.





### 3.2 Estándar de codificación empleado

Con el objetivo de lograr una estandarización en la implementación de la herramienta informática se decide aplicar el estilo de escritura empleado en estándares de codificación CamelCase, específicamente la variante LowerCamelCase, que se caracteriza porque las palabras van unidas entre sí sin espacios; con la peculiaridad de que la primera letra de cada término se encuentra en minúscula para hacer más legible el conjunto, esto cual facilitará su posterior soporte y mantenimiento, así como una mejor lectura, comprensión del código. A continuación, se describe a grandes rasgos las convenciones de nomenclatura empleadas.

#### General:

- ✓ Se exceptúan el uso de las tildes y la letra ñ.
- ✓ En todo momento se utilizarán nombres que sean claros, concretos y libres de ambigüedades.
- ✓ El nombre de todas las variables, métodos y clases comenzarán con letra minúscula. Ejemplo: `def get_context_data (self, id=False, **kwargs):`.

#### Indentación:

- ✓ El contenido siempre se indentará con *tabs*, nunca utilizando espacios en blanco.

#### Clases:

- ✓ El nombre de las clases comenzará con minúscula, si este está compuesto por varias palabras, la primera palabra interna comenzará con minúscula también, pero con la peculiaridad que las demás comenzarán con mayúscula.
- ✓ Ejemplo: `class logsCriticidadReport (PDFTemplateView):`.
- ✓ Intentar mantener los nombres de las clases descriptivos y simples. Usar palabras completas, evitar acrónimos y abreviaturas, a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML.

#### Nombre de variables:

- ✓ No se utilizarán nombres de variables que puedan ser ambiguos.
- ✓ Se procurará evitar asignar nombres sin sentido a variables temporales, por ejemplo: temp, i, tmp.



### **3.3 Validación**

Una vez concluida la disciplina análisis y diseño se comenzará con la validación, donde a partir de los resultados obtenidos anteriormente se validará y seguidamente se implementará el sistema en términos de componentes, ficheros, código fuente, scripts ejecutables y similares. Con el objetivo de certificar esta disciplina se realizó la validación de los resultados obtenidos con el uso de métricas.

Las métricas para el modelado del diseño cuantifican los atributos de diseño de manera tal que le permiten al ingeniero de software evaluar la calidad del diseño. Estas métricas de modelado del diseño incluyen: métricas arquitectónicas, métricas al nivel de componentes, métricas de diseño de interfaz y métricas especializadas en el diseño orientado a objeto(Presman, 2007).

#### **3.3.1 Validación del diseño del sistema**

Para comprobar la calidad y fiabilidad de la herramienta ‘‘Análisis de criticidad’’, se emplearon las métricas basadas en clases: Tamaño Operacional de Clase (TOC) y Relaciones entre Clases (RC), permitiendo el resultado de ambas métricas validar el diseño propuesto en la investigación. A continuación, se muestra un resumen de los resultados de la aplicación de ambas métricas.

##### **3.3.1.1 Tamaño Operacional de Clases (TOC)**

La métrica TOC es una métrica especializada en diseño orientado a objeto, midiendo características de clases, además de las correspondientes a comunicación y colaboración. Está dado por el número de métodos y atributos asignados a una clase para evaluar los siguientes atributos de calidad: responsabilidad, complejidad de implementación y la reutilización de las clases del diseño. Es importante destacar que para esta métrica, la responsabilidad y la complejidad son inversamente proporcionales a la reutilización, por lo que a mayor responsabilidad y complejidad de implementación de una clase, menor será su nivel de reutilización(Lorenz, 1994). El tamaño operacional de una clase se puede determinar empleando medidas para saber el número total de operaciones que contiene.

A continuación, se muestra el resultado de la aplicación de la métrica TOC en las 108 clases que juegan un papel fundamental en los procesos principales del sistema informático.

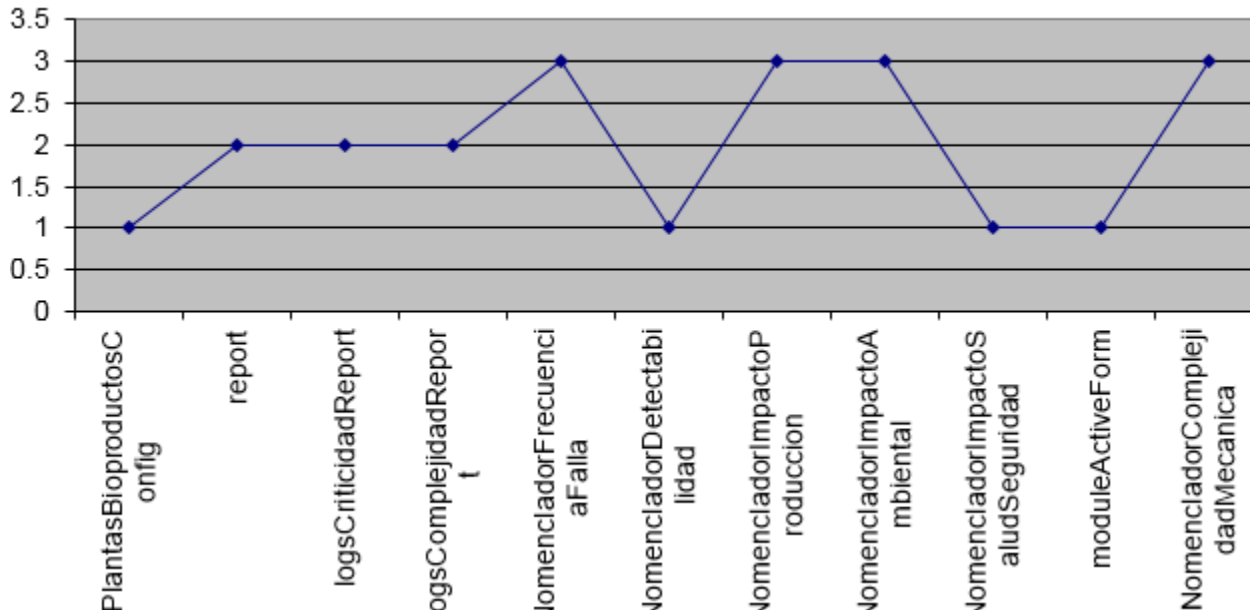


Figura 14: Resumen de la cantidad de procedimiento por clases.

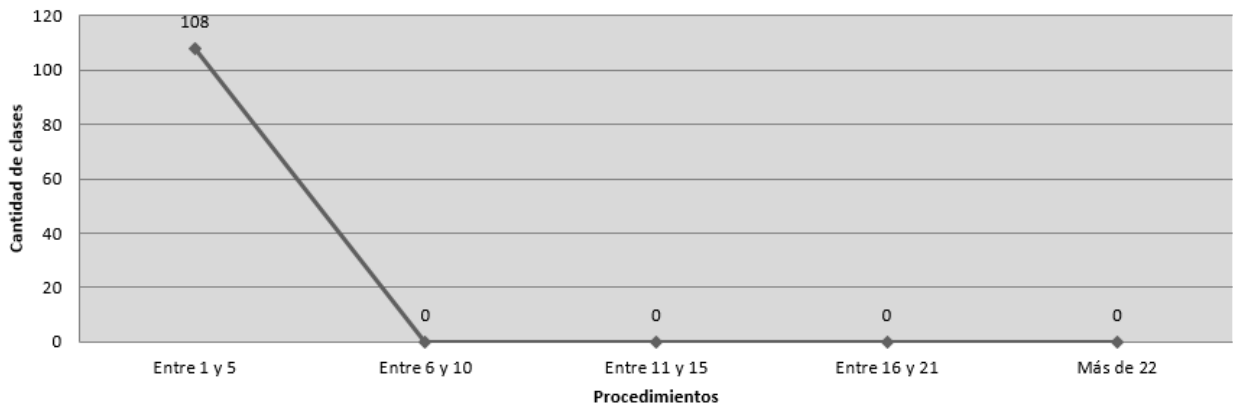


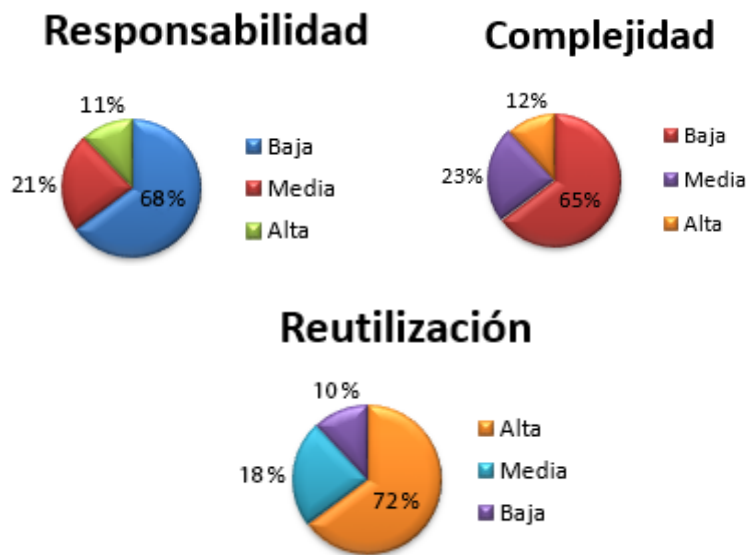
Figura 15: Resumen de cantidad de procedimientos.

Aplicando como umbral un promedio de 1.9537 procedimientos (211 procedimientos en total /108 clases existentes), se obtienen los valores de los atributos de calidad: responsabilidad de las clases, complejidad al implementar las mismas, así como sus niveles de reutilización.



**Tabla 3:** Valores de las variables de calidad: Responsabilidad, Complejidad y Reutilización.

Nivel	Responsabilidad		Complejidad		Reutilización	
	Cantidad de Clases	Promedio	Cantidad de Clases	Promedio	Cantidad de Clases	Promedio
Baja	75	67.814	70	64.814	81	72.8148
Media	15	21.148	25	23.148	12	18.1481
Alta	10	11.038	13	12.037	7	9.0371

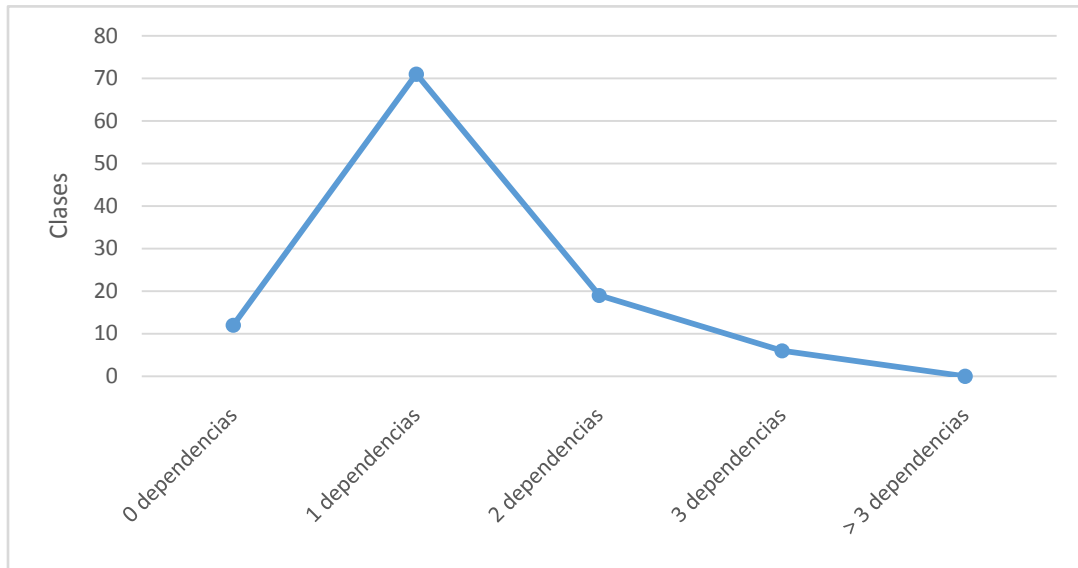


**Figura 16:** Resultado de las variables: Responsabilidad, Complejidad y Reutilización.

Luego de aplicada la métrica TOC se observa que las clases del diseño de la herramienta no se encuentran sobrecargadas en cuanto a responsabilidades, y el nivel de complejidad de las mismas no es muy alto, lo que favorece en gran medida la reutilización de estas.

### 3.3.1.2 Relaciones entre Clases (RC)

La métrica RC permite evaluar el acoplamiento, la complejidad de mantenimiento, la reutilización y la cantidad de pruebas de unidad necesarias para probar una clase teniendo en cuenta las relaciones existentes entre ellas. Luego de aplicar la métrica RC, se arrojaron los siguientes resultados:



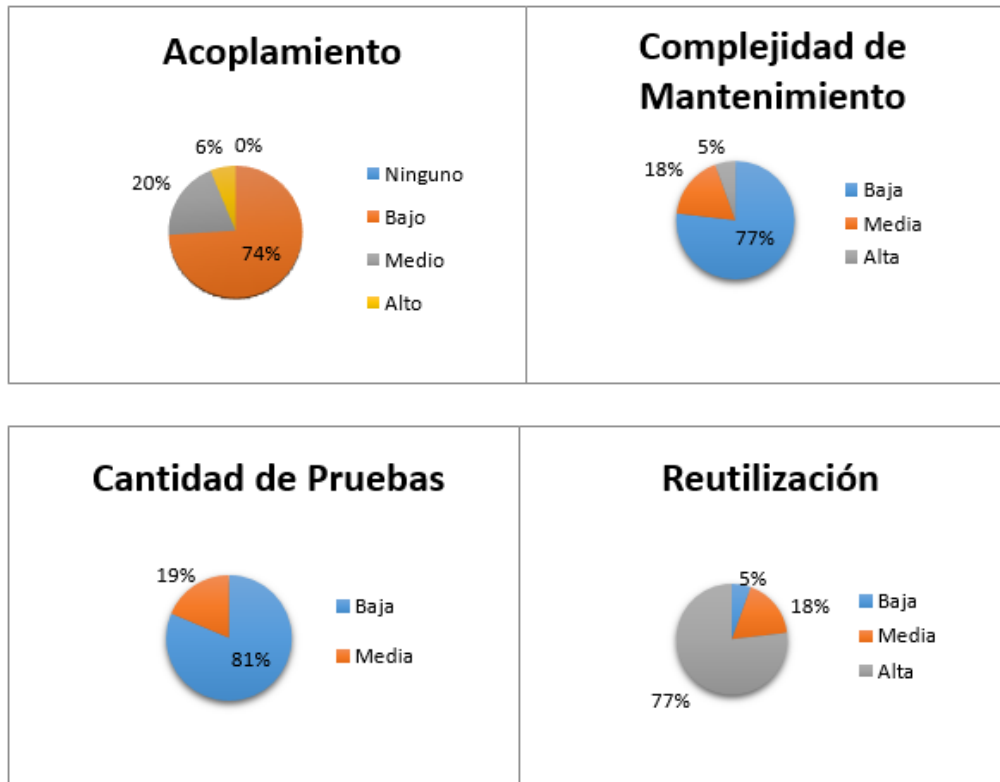
**Figura 17:** Resumen de relaciones de uso.

Luego de calcular los valores de las variables de calidad: acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas, aplicando como umbral un promedio de 2.1559 asociaciones de uso por clase; obteniendo los siguientes resultados:

**Tabla 4:** Valores de las variables: Acoplamiento, Complejidad, Reutilización y Cantidad de Pruebas

Nivel	Acoplamiento		Complejidad del Mantenimiento		Reutilización		Cantidad de Pruebas	
	Cantidad de Clases	Promedio	Cantidad de Clases	Promedio	Cantidad de Clases	Promedio	Cantidad de Clases	Promedio
<b>Ninguno</b>	0	0	-	-	-	-	-	-
<b>Bajo</b>	71	65.740	83	76.851	6	5.555	83	76.851
<b>Medio</b>	19	17.592	19	17.592	19	17.592	25	23.148
<b>Alto</b>	6	5.555	6	5.555	83	76.851	0	0

Una vez aplicada la métrica RC y teniendo en cuenta el umbral definido para validar el diseño, se obtiene como resultado que las clases promueven el bajo acoplamiento; la complejidad del mantenimiento y el indicador de la cantidad de pruebas son bajas, y, en consecuencia, el grado de reutilización es alto.



**Figura 18:** Resultado de las variables Acoplamiento, Complejidad del Mantenimiento, Reutilización y Cantidad de Pruebas.

En sentido general, los resultados obtenidos de la aplicación de las métricas TOC y RC demuestran que el diseño de la solución no es complejo, las clases no se encuentran sobrecargadas en cuanto a responsabilidades, el nivel de complejidad de las mismas no es muy alto, lo que favorece en gran medida su reutilización. Además, las clases presentan bajo acoplamiento y un alto grado de reutilización. Esto se valora de positivo pues existen muy pocas relaciones de uso entre los nuevos elementos que se adicionan, lo que trae como consecuencia que al existir cambios en el sistema la repercusión en el resto de los elementos es mínima. El bajo acoplamiento de elementos favorece a la reutilización, la cual es alta en un 77% de las clases del componente. Por otra parte, la complejidad de mantenimiento, de implementación, así como la cantidad de pruebas arrojan valores mayormente bajos, por tanto, se requiere de poco esfuerzo en estas tareas. Teniendo en cuenta los resultados obtenidos y lo que representan cada uno para la solución, es posible afirmar que el diseño propuesto tributa al desarrollo de una aplicación con calidad.



### 3.3.1.3 Adaptabilidad del diseño de la aplicación

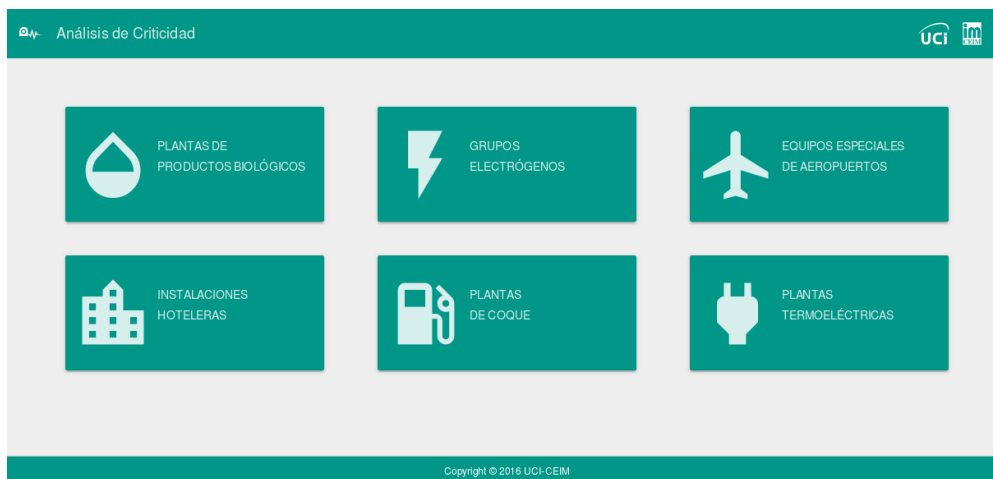
Con el fin de validar que el sistema propuesto como solución cuenta con un diseño adaptativo se instaló en una máquina destinada como servidora, luego se accedió a la misma desde diferentes computadoras y terminales con distintas configuraciones de pantalla (resoluciones). A través de esta prueba se pudo observar que el sistema se adaptaba a las distintas resoluciones de pantalla, desde un portátil, una computadora de escritorio hasta un smartphone. A continuación, se presentan imágenes tomadas al sistema que satisfacen lo anteriormente descrito.



**Figura 19:** Interfaz principal del sistema *Análisis de Criticidad* en una pantalla con resolución de 320x480 (pantalla de un smartphone).



**Figura 20:** Interfaz principal del sistema *Análisis de Criticidad* en una pantalla con resolución de 800x600.



*Figura 21: Interfaz principal del sistema Análisis de Criticidad en una pantalla con resolución de 1024x768.*

### 3.4 Pruebas

Una vez terminada la implementación del producto que se requiere es necesario realizarle pruebas, con el objetivo de detectar errores en la aplicación y la documentación; este proceso resulta de gran importancia porque da una medida de la calidad del mismo, siempre que se lleve a cabo de la forma correcta.

Las pruebas son los procesos que permiten verificar y revelar la calidad de un producto de *software*. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un *software*. En este proceso se ejecutan pruebas dirigidas al sistema de *software* en su totalidad, con el objetivo de medir el grado en que el mismo cumple con los requerimientos.

#### 3.2.1 Pruebas Internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción incluyendo tanto las construcciones internas como las intermedias, así como las versiones finales a ser liberadas. Se genera el artefacto de prueba diseño de casos de prueba.





### 3.3.2.1 Pruebas de Funcionalidad

Las pruebas funcionales, también denominadas pruebas de comportamiento o de caja negra se centran en los requisitos funcionales del software. O sea, la prueba de caja negra garantiza al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Su principal objetivo es comprobar que las funcionalidades de la aplicación se realizan de forma correcta y responden a las necesidades del cliente, y permiten descubrir otra clase de errores tales como: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a bases de datos externas, errores de rendimiento, errores de inicialización y de terminación(Pressman, 2010), apoyándose en los casos de pruebas diseñados para cada funcionalidad. Esto se puede evidenciar en el producto de trabajo diseños de casos de prueba, generado en la disciplina pruebas internas de la etapa de ejecución de la metodología de desarrollo de *software* empleada.

La partición equivalente según Roger S. Pressman es una técnica de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Esta se dirige a la definición de casos de prueba que descubran clases de errores, para reducir el número total de casos de prueba que hay que desarrollar. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada(Pressman, 2010).

Las pruebas se realizan a partir de una matriz de datos donde 'V' indica válido, 'I' inválido y 'NA' indica que no es necesario proporcionar un valor del dato, ya que es irrelevante. A continuación, se muestra la descripción de las variables que representan valores de entrada de datos para los casos de prueba aplicados a los casos de usos AdicionareImportar.



**Tabla 5:** Descripción del caso de prueba para el caso de uso “Adicionar”.

Escenario	Descripción	Nombre	Código	Área	Respuesta del sistema	Flujo central
EC 1.1 Adicionar activo	Se adicionan nuevos activos correctamente	V	V	V	El sistema almacena los cambios y lo lista	Se hace clic en la funcionalidad Adicionar localizada en la barra de menú -Se insertan los parámetros en los campos mostrados. -Se presiona el botón Aceptar -Se validan los datos.
		name	code	area		
EC 1.2 Adicionar activo dejando campos vacíos	No se insertan parámetros en los campos mostrados al adicionar nuevos activos		V	V	El sistema resalta el nombre del campo que está vacío indicando que es obligatorio y que se debe insertar algún parámetro	Se hace clic en la funcionalidad Adicionar localizada en la barra de menú -Se insertan los parámetros en algunos de los campos mostrados. -Se presiona el botón Aceptar
		name	code	area		
		V		V		
		name	code	area		
		V	V			
EC 1.2 Adicionar activo insertando caracteres inválidos	Se insertan parámetros inválidos en los campos mostrados al adicionar nuevos activos	I	N/A	I	El sistema resalta el nombre del campo que contiene caracteres inválidos indicando que debe corregirse	Se hace clic en la funcionalidad Adicionar localizada en la barra de menú -Se insertan los parámetros en los campos mostrados. -Se presiona el botón Aceptar
		name	code	area		
		V	N/A	I		
		name	code	area		
		I	N/A	V		
name	code	area				

**Tabla 6:** Descripción de las variables correspondiente al caso de prueba para el caso de uso “Adicionar”.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No	Caracteres alfanuméricos
2	Código	Campo de texto	No	Caracteres alfanuméricos y especiales
3	Area	Campo de texto	No	Caracteres alfanuméricos

**Tabla 7:** Caso de prueba para el caso de uso “Reporte”

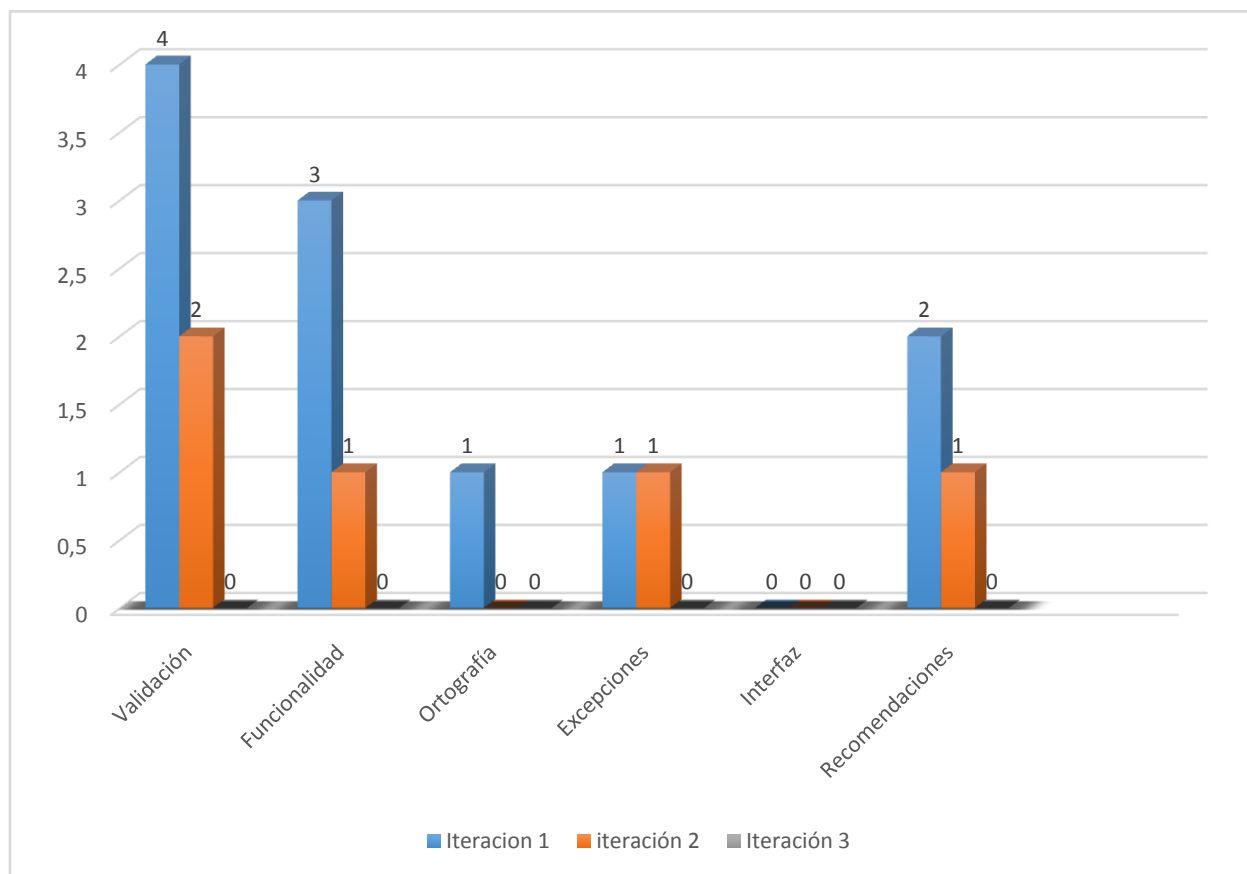
Escenario	Descripción	Área	Respuesta del sistema	Flujo central
EC 1.1 Exportar Reporte	Se exporta el reporte de criticidad de manera correcta	V area	El sistema almacena el reporte en formato PDF en la ruta que el usuario defina	Se hace clic en la funcionalidad Reporte localizada en la barra de menú -Se seleccionan las áreas por las que se desea filtrar el reporte. -Se presiona el botón Aceptar -Se validan los datos.
EC 1.2 Exportar Reporte sin filtrar áreas	No se filtran areas al exportar el reporte de criticidad	I area	El sistema muestra una notificación al usuario indicándole que debe filtrar por al menos un área para poder exportar el reporte	Se hace clic en la funcionalidad Reporte localizada en la barra de menú -No se seleccionan areas para filtrar el reporte. -Se presiona el botón Aceptar

**Tabla 8:** Descripción de las variables correspondiente al caso de prueba para el caso de uso “Reporte”

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Area	CheckBox	No	CheckBox para seleccionar

Para comprobar la calidad de la herramienta “Análisis de Criticidad” se realizaron 3 iteraciones por parte del grupo de calidad perteneciente a la Subdirección del Centro de Informatización de Entidades (CEIGE). Se encontraron un total de once no conformidades (NC) en la primera iteración, clasificándose cuatro de ellas de validación, tres de funcionalidad, una de ortografía, una de excepciones y dos recomendaciones. En la segunda iteración se detectaron un total de cinco (NC), donde dos de ellas fueron de validación, una de funcionalidad, una de excepciones y una recomendación. Finalmente, en la tercera iteración no se detectaron NC.

En la siguiente figura se muestran las iteraciones realizadas y la cantidad de NC encontradas en cada iteración.



**Figura 22:** Relación de las NC, aplicando las pruebas de Caja Negra

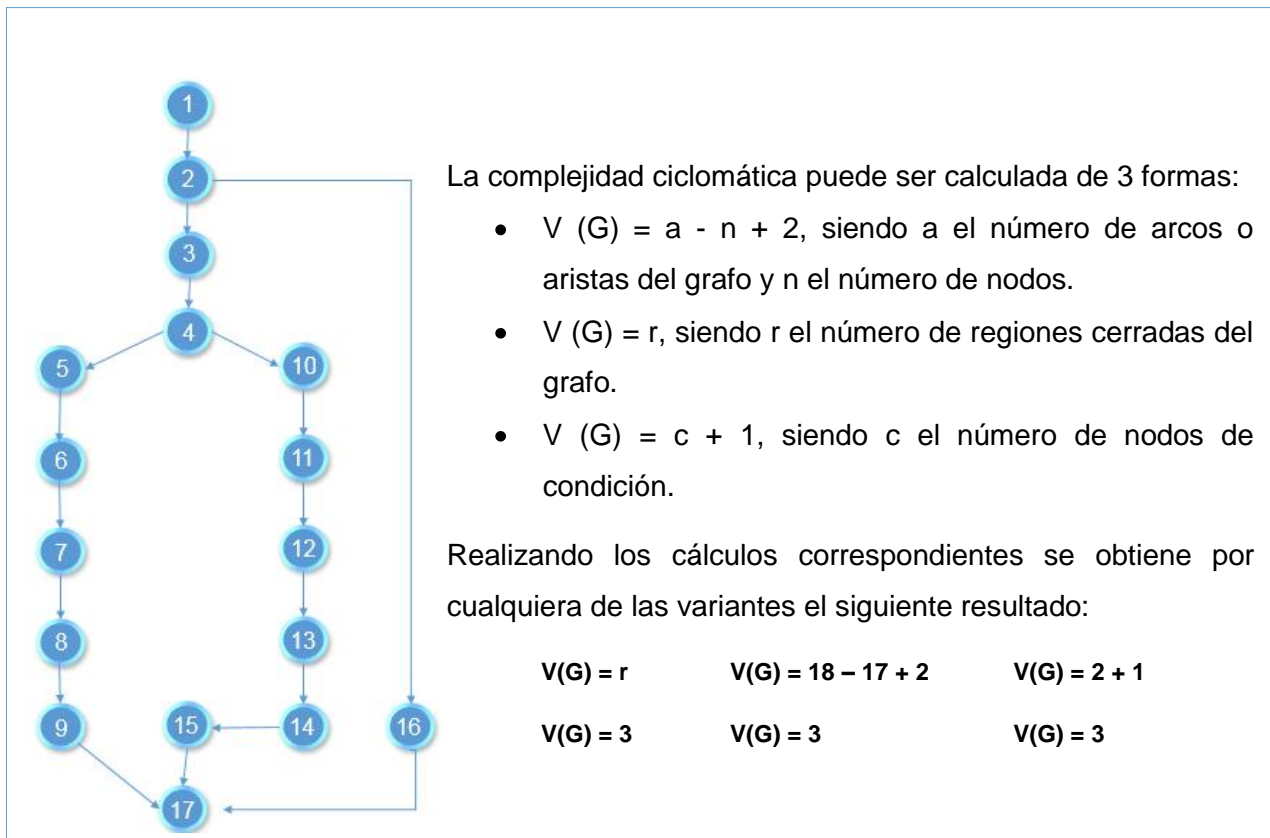
Todas estas no conformidades encontradas por parte del grupo de calidad en las iteraciones de la revisión de la herramienta fueron corregidas satisfactoriamente y en el tiempo establecido para hacerlo, por lo que se emitió un acta de liberación de calidad del producto por parte del centro CEIGE, dicha acta se encuentra plasmada en la sección Anexos.

### 3.2.2 Pruebas unitarias

Para comprobar que cada sentencia de código se ejecuta al menos una vez, se realizaron pruebas al código de las funcionalidades más complejas desde el punto de vista de la programación en cada uno de los módulos. Para ello se empleó el método de caja blanca con la técnica del *Camino Básico*; a partir de la obtención de la medida de la complejidad de un procedimiento o algoritmo y la obtención de un conjunto básico de caminos de ejecución de este, los cuales son utilizados para obtener los casos de prueba.



Seguidamente se muestra el proceso de pruebas realizado a el método *addActive* perteneciente al módulo Plantas Termoeléctricas, para obtener los casos de prueba a partir de la técnica seleccionada se debe construir el grafo de flujo correspondiente al código de la funcionalidad. Luego se determina la complejidad ciclomática  $V(G)$  del grafo resultante, la cual es un indicador del número de caminos independientes que existen en un grafo, es decir, es cualquier camino dentro del código que introduce por lo menos un nuevo conjunto de sentencias de proceso o una nueva condición (Rojas, 2010).



**Figura 23:** Grafo de flujo del código de la función *addActive* y cálculo de su complejidad ciclomática.

Por lo que el conjunto de caminos básico sería:

*Camino Básico 1:* 1-2-16-17

*Camino Básico 2:* 1-2-3-4-5-6-7-8-9-17

*Camino Básico 3:* 1-2-3-4-10-11-12-13-14-15-17



Una vez definidos los caminos básicos del flujo se pasa a ejecutar los casos de prueba para cada uno de estos. Para definir los casos de prueba es necesario tener en cuenta:

- ✓ **Descripción:** se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.
- ✓ **Condición de ejecución:** se especifica cada parámetro para que cumpla una condición deseada y así ver el funcionamiento del procedimiento.
- ✓ **Entrada:** se muestran los parámetros que serán la entrada al procedimiento.
- ✓ **Resultados Esperados:** se expone el resultado esperado que debe devolver el procedimiento después de efectuado el caso de prueba.

A continuación, se muestra el resultado de las pruebas aplicadas de la función **addActive**, para lo que se determinó escoger el camino básico 2.

```
Camino Básico 2: def addActive(request):  
if request.method == 'POST':  
    formulario = moduleActiveForm(request.POST)  
Si el formulario es válido ()  
    name = valor 1  
code = valor 2  
area = valor 3  
Guardar valores del Formulario en la base de datos ()  
Retornar "Notificación de operación realizada con éxito".
```

Con la realización de estas pruebas se logró garantizar que todas las sentencias del programa fueran ejecutadas al menos una vez.

### 3.3 Caso de estudio

Para la evaluación del correcto funcionamiento de la herramienta "Análisis de Criticidad" se evaluó un caso de estudio real descrito en la tesis de maestría (Pérez, 2008), donde se definen un conjunto de activos con sus respectivos índices de criticidad calculados previamente, así como los parámetros y sus ponderaciones seleccionadas para obtener dichos índices de criticidad y complejidad. A continuación, se presenta una muestra de 15 de un total de 33 activos donde se refleja cada uno de los parámetros y los valores de sus ponderaciones, tomado del caso de estudio.



Ponderaciones por equipos												
No.	Nombre del equipo	Área	Severidad			Frec. Falla	Complejidad			Detectabilidad	I.Crt.	I.Com.
			I.A.	I.S.	I.P.		C.P.	C.M.	C.U.			
1.-	Zaranda	Laboratorio	0	0	5	2	1	3	3	3	12	7
2.-	Fermentadores de inóculo	Laboratorio	5	5	5	2	3	3	3	3	36	9
3.-	Fermentadores de producción	Producción	25	25	10	3	5	5	3	3	216	13
4.-	Tanques	Producción	10	5	3	1	1	1	3	10	72	5
5.-	Bombas de trasiego	Producción	5	5	5	3	1	3	3	7	126	7
6.-	Bombas dosificadoras	Producción	5	5	3	2	1	3	3	7	72.8	7
7.-	Tanques	Recobrado	10	0	3	1	1	3	3	7	36.4	7
8.-	Centrífuga de discos	Recobrado	25	25	10	4	5	5	3	3	288	13
9.-	Centrífuga alta resolución	Recobrado	25	25	10	4	5	5	3	5	480	13
10.-	Bombas de trasiego (Centrífugas)	Recobrado	5	5	3	2	1	3	3	7	72.8	7
11.-	Secador	Terminación	25	25	10	3	5	5	3	5	360	13
12.-	Tanques	Mat. Primas	5	0	3	1	1	3	3	7	22.4	7
13.-	Reactores	Mat. Primas	10	25	5	1	3	3	0	7	112	6
14.-	Bombas Centrífugas	Mat. Primas	5	0	3	2	1	1	0	7	44.8	2
15.-	Compresores	Sumin. Aire	0	10	10	2	1	5	0	7	112	6

*Figura 24: Ponderaciones y resultados obtenidos en la evaluación del análisis de criticidad y complejidad en el caso de estudio.*

Estos activos fueron insertados en el sistema propuesto como solución, con el objetivo de realizarles el análisis de criticidad y complejidad a partir de cada uno de los valores de las ponderaciones para los parámetros que se evalúan, y de esta forma comparar los resultados obtenidos por la aplicación con los del caso de estudio anteriormente referenciado. Comprobando de esta forma que los valores arrojados por la herramienta implementada son iguales a los obtenidos en el caso de estudio (Ver Figuras 18 y 19). A continuación, se ilustran los resultados obtenidos por el sistema propuesto.



Análisis de Criticidad >> Plantas de Productos Biológicos

UCI

Importar Activos Adicionar Activo Calcular Historial Graficar Eliminar Valores Generar Reporte

Mostrar 25 elementos Buscar Activo

No.	Código	Nombre	Área	Criticidad	Complejidad	
1	COD001	Zaranda	Laboratorio	12.0	7.0	
2	COD002	Fermentadores de inóculo	Laboratorio	36.0	9.0	
3	COD003	Fermentadores de producción	Producción	216.0	13.0	
4	COD004	Tanques	Producción	72.0	5.0	
5	COD005	Bombas de trasiego	Producción	126.0	7.0	
6	COD006	Bombas dosificadoras	Producción	72.8	7.0	
7	COD007	Tanques	Recobrado	36.4	7.0	
8	COD008	Centrifuga de discos	Recobrado	288.0	13.0	
9	COD009	Centrifuga alta resolución	Recobrado	480.0	13.0	
10	COD010	Bombas de trasiego	Recobrado	72.8	7.0	
11	COD011	Secador	Terminación	360.0	13.0	
12	COD012	Tanques	Materias Primas	22.4	7.0	
13	COD013	Reactores	Materias Primas	112.0	6.0	
14	COD014	Bombas Centrifugas	Materias Primas	44.8	2.0	
15	COD015	Compresores	Suministro de Aire	112.0	6.0	

Copyright © 2016 UCI - CEIM

*Figura 25: Resultados obtenidos del análisis de criticidad y complejidad en la herramienta propuesta como solución.*





### 3.4 Conclusiones Parciales

En este capítulo se realizó la implementación y validación de la propuesta de solución arrojando resultados importantes tales como: se validaron las fases de desarrollo abarcada durante el proceso de implementación de la herramienta informática con buenas prácticas, permitiendo obtener un producto de calidad; se realizaron pruebas de *software* que facilitaron identificar y solucionar las deficiencias detectadas en el sistema proyectando resultados satisfactorios en el desarrollo del sistema como producto final y solución al problema planteado.

La definición y utilización del estándar de codificación *CamelCase*, específicamente la variante *LowerCamelCase*, en la implementación del sistema propuesto permitió el desarrollo de los componentes con un alto grado de legibilidad; lo que provee una guía para el mantenimiento y actualización del sistema, con código claro y bien documentado. Las pruebas realizadas como parte del proceso de desarrollo arrojaron como resultado que tanto el diseño como la implementación de la aplicación presentan un nivel de complejidad relativamente bajo, y un alto nivel de reutilización, lo cual favorece la mantenibilidad.

Por último, se realizó una valoración del comportamiento de las variables que forman parte del problema de la investigación, demostrando que, con el sistema desarrollado, la realización de los análisis de criticidad se logró mejorar, evidenciándose en una mejor toma de decisiones en la Ingeniería del Mantenimiento, garantizando además la precisión de los resultados obtenidos.

## CONCLUSIONES

Con la realización de este trabajo se cierra un período de desafíos y gratitudes asociadas a la superación de un número importante de insatisfacciones a la hora de realizar análisis de criticidades en la Ingeniería de Mantenimiento y con ello en la toma de decisiones.

Sobre la base del análisis, interpretación y sistematización de las investigaciones teóricas y empíricas, a continuación, se presentan las siguientes conclusiones de la investigación:

1. Se realizó el marco teórico de la investigación donde se establecieron las tendencias marcadas en cuanto al desarrollo de *software* para realizar análisis de criticidad y sus modelos matemáticos, lo que permitió fundamentar la necesidad de desarrollar una herramienta informática para realizar estos análisis.
2. Se analizaron los modelos de criticidad existentes, permitiendo adquirir el conocimiento necesario acerca de los algoritmos matemáticos empleados, para realizar el análisis de criticidad utilizado en la toma de decisiones de los ingenieros en mantenimiento.
3. El diseño de la herramienta fue validado a través de las métricas TOC y RC, arrojando como resultado positivo que el sistema propuesto no es complejo, que las clases presentan bajo acoplamiento y un alto grado de reutilización.
4. La implementación de la herramienta permitió mejorar los análisis de criticidad y con ello la toma de decisiones de los ingenieros en mantenimiento.
5. Para validar la herramienta se realizaron pruebas de caja blanca, caja negra y prueba de correlación de resultados, comprobándose el correcto funcionamiento de la misma, evidenciándose mediante un acta de liberación redactada por el equipo de calidad del centro CEIGE.

## **RECOMENDACIONES**

El autor del presente trabajo recomienda:

1. Divulgar los principales resultados de la investigación, a partir de la presentación y aplicación de la propuesta, a instituciones que realicen el cálculo de criticidad en Cuba y extender su uso en las mismas.
2. Integrar a la herramienta futuros modelos de criticidad que se validen con posterioridad.

## REFERENCIAS BIBLIOGRÁFICAS

- Alvarez, Miguel Angel. 2003.***DesarrolloWeb.com*. [En línea] 2003. [Citado el: 5 de Noviembre de 2015.] <http://www.desarrolloweb.com/articulos/1325.php>.
- Arizaca, Roberto E. 2011.***Artefacto: Diagrama de Componentes*. La Paz, Bolivia. : s.n., 2011.
- Barraza, Alberto. 2014.***Slideshare.net*. [En línea] 2014. [Citado el: 16 de Noviembre de 2015.] <http://es.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml>.
- Bennett, James. 2015.** Djangoproject.com. *The Web Framework for perfectionists with deadlines / Django*. [En línea] 2015. [Citado el: 18 de Noviembre de 2015.] <https://www.djangoproject.com/>.
- Calzada, Ing. María Bárbara Hourné. 2010.***Análisis de criticidad personalizado de las Plantas Eléctricas de Grupos Electrógenos de la tecnología fuel oil en Cuba*. 2010.
- Camarillo, Julián. 2016.***Debitoor*. [En línea] 2016. [Citado el: 20 de Noviembre de 2015.] <https://debitoor.es/glosario/definicion-de-activo>.
- Carballo, Ing. Michel Cúcalo. 2006.** Formulación y Validación de una expresión para el análisis de Criticidad del Parque de Equipos del Aeropuerto. Tutor Ing. Guido Enrique Orta García. Ciudad de la Habana, Habana, Cuba : s.n., Junio de 2006.
- Díaz González, José Daniel y Sánchez Pedroso, Victoria Caridad. 2014.** Módulo para el cálculo de criticidad de activos en Instalaciones Hoteleras, Aeropuertos y Plantas de Coque. La Habana : s.n., 2014.
- Díaz, Carlos Alberto. 2011.***Estudio sobre las características de la gestión del mantenimiento en las plantas de producción productos biológicos*. 2011.
- Elcano, Sebastián. 2014.** PyCharm el IDE de Python. *Hipertextual.com*. [En línea] 2014. [Citado el: 26 de Noviembre de 2015.] <http://hipertextual.com/archivo/2014/06/pycharm-ide-python/>.
- Fernández, Ing. Rosendo Alfonso. 2013.***Obtención y validación de un Modelo de Criticidad para los equipos y sistemas tecnológicos en Plantas de Coque*. 2013.
- Fernández, Rafael J. 2011.***Modelo de Datos*. 2011.
- Fowler, Martin. 2003.** Patterns of Enterprise Application Architecture. [En línea] 2003. [Citado el: 1 de Diciembre de 2015.] <https://books.google.com/cu/books?id=vqTfNFDzzdIC&printsec=frontcover&dq=inauthor:%22Martin+Fowler%22&hl=es&sa=X&ved=0CE4Q6AEwBmoVChMIIm6K2stSExgIVyC2MCh0kCwDp#v=onepage&q&f=false>. ISBN 9780321127426..
- Gamma, Richard. 2009.***Design Patterns: Elements of Reusable Object-Oriented Software*. s.l. : Pearson Education, 2009. ISBN 9780321700698.

**García, Ing. Manuel Toledo. 2015.** Obtención y Aplicación de un Modelo de Criticidad para los equipos y sistemas tecnológicos en Plantas Termoeléctricas. Tutor MSc. Armando Díaz Concepción. Cienfuegos, Cienfuegos, Cuba : s.n., Marzo de 2015.

**Gárciga, Ing. Alfredo Barrios. 2005.** Modelo de Análisis de Criticidad de los subsistemas objetos de mantenimiento en Instalaciones Hoteleras. Tutor Dr. Alfredo del Castillo. Ciudad de la Habana, Habana, Cuba : s.n., Julio de 2005.

**Gehrke, Johannes. 2012.** *Monografias.com*. [En línea] 2012. [Citado el: 3 de Diciembre de 2015.] <http://www.monografias.com/trabajos-pdf2/sistema-gestion-base-datos-postgresql/sistema-gestion-base-datos-postgresql.pdf>.

**Graham, Tim. 2013.** *Django en Español*. [En línea] 2013. [Citado el: 6 de Diciembre de 2015.] <http://django.es/>.

**Grosso, Andrés. 2011.** Patrones-Grasp. [En línea] 2011. [Citado el: 10 de Diciembre de 2015.] <http://www.practicadesoftware.com.ar/2011/03/patrones-grasp/>.

**Gudgin, Martin. 2007.** SOAP version 1.2 Part 1: Messaging Framework. W3. [En línea] 2007. [Citado el: 14 de Diciembre de 2015.] <http://www.w3.org/TR/soap12-part1/>.

**Jacobson, Ivar. 2000.** *El proceso unificado de desarrollo de software*. s.l. : Addison Wesley Reading, 2000.

**Jiménez, Leandro. 2011.** Patrones Grasp. [En línea] 2011. [Citado el: 21 de Diciembre de 2015.] <http://www.buenastareas.com/ensayos/Patrones-Grasp/1896730.html>.

**Justensen, Alessandro. 2015.** *Definicion*. [En línea] 2015. [Citado el: 10 de Enero de 2016.] <http://definicion.mx/proceso/>.

**Larman, Craig. 2003.** *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado*. s.l. : Pearson Education, 2003. 9788420534381.

—. 2003. *UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado*. s.l. : Pearson Educación, 2003. ISBN 9788420534381.

**Lau, Ricardo. 2011.** Entorno de Desarrollo Integrado (IDE). *Vigo: Unión Webmaster Libres*. [En línea] 2011. [Citado el: 14 de Enero de 2016.] <https://sites.google.com/site/vigomiciudad/otras-cosa-interesantes/ide-entornos-de-desarrollo-integrado>.

**Lobos, María Elena de los. 2005.** *Mialxmail*. [En línea] 2005. [Citado el: 29 de Noviembre de 2015.] <http://www.mailxmail.com/curso-aprende-programar/tipos-estructuras-programacion-estructuras-basicas-secuencial>.

**Lorenz, Lorenz y Kidd. 1994.** *Métricas para la validación del diseño*. 1994.

**Martínez, Carolina. 2012.** Ingeniería de Software 1. 2012.

- Martínez, Jorge Enrique. 2008.** Asociación Argentina de Materiales (SAM). *Materiales-sam.org*. [En línea] 2008. [Citado el: 26 de Enero de 2016.] [http://www.materiales-sam.org.ar/sitio/revista/1\\_2011/p29-42.pdf](http://www.materiales-sam.org.ar/sitio/revista/1_2011/p29-42.pdf).
- McKay, Andy. 2011.***Regoremor*. [En línea] 2011. [Citado el: 19 de Enero de 2016.] <http://www.regoremor.com/desarrollo-web/css/frameworks-de-css-para-el-desarrollo-web/>.
- Mendoza, Ing. Rosendo Huerta. 2004.** El análisis de criticidad, una Metodología para mejorar la Confiabilidad Operacional. Venezuela, Venezuela : Publicación Periódica del Club de Mantenimiento, 2004.
- Montero, Sergio Infante. 2012.** LibrosWeb. *librosweb.es*. [En línea] Abril de 2012. [Citado el: 29 de Enero de 2016.] [http://librosweb.es/5\\_2\\_ El patrón de diseño MTV \(El libro de Django 1\\_0\).htm](http://librosweb.es/5_2_ El patrón de diseño MTV (El libro de Django 1_0).htm).
- Nieto, Moisés Muñoz. 2015.***Diseño y Arquitectura de Django*. 2015.
- Osorio, Msc. Ing. David Cabrera. 2005.** Análisis de Criticidad de Equipos de Subestaciones y Componentes del Sistema de Transmision de la TDE. Tutor Dr. Alfredo del Castillo. Diciembre de 2005.
- Pérez, Ing. Frank Rodríguez. 2008.** Obtención y validación de un modelo para el análisis de criticidad de equipos en Plantas de Produccion de Productos Biológicos. Tutor MSc. Ing. Armando Díaz Concepción. CEIM/ISPJAE. Ciudad de la Habana, Habana, Cuba : s.n., 2008.
- Pérez, Isaías Carrillo. 2014.***Metodologías de Desarrollo*. [En línea] 2014. [Citado el: 22 de Diciembre de 2015.] <https://es.scribd.com/doc/54060274/Metodologias-de-desarrollo>.
- Pérez, Navarrete. 2012.***Gestión e Ingeniería Integral del Mantenimiento*. 2012.
- Pettus, Christophe. 2014.***Maestros del Web*. [En línea] 2014. [Citado el: 30 de Noviembre de 2015.] <http://www.maestrosdelweb.com/comparacion-frameworks-javascript/>.
- Presman, Roger S. 2007.***Software Engineering: A Practitioner's Approach*. s.l. : Palgrave Macmillan, 2007.
- Pressman, Roger S. 2010.***Software Engineering. Septima edición*. Nueva York : s.n., 2010.
- Procida, Daniele. 2013.***Capacity*. [En línea] 2013. [Citado el: 13 de Enero de 2016.] <http://blog.capacityacademy.com/2013/03/16/jquery-que-es-origenes-ventajas-desventajas/>.
- Ramakrishhnan, Raghu. 2003.***Database Management Systems*. New York : McGraw-Hill, 2003.
- Riehle, Dirk. 2015.***Framework design*. [En línea] 2015. [Citado el: 27 de Febrero de 2016.] <http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf>.
- Riveros, Ing. Leonardo Montaña. 2006.** Diseño de un Sistema de Mantenimiento con base en Análisis de Criticidad y Análisis de ,modos y efectos de falla en la planta de Coque de fabricación

primaria en la Empresa Acerías Paz del Río S.A. Universidad Pedagógica y Tecnológica. Facultad Seccional. Escuela de Ingeniería Electromecánica-Duitama, Colombia, Cuba : s.n., 2006.

**Rojas, Emilio. 2010.***Pruebas.* s.l. : [Disponible en: <http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node26.html>, 2010.

**Romay, Wilmer. 2013.***Scribd.com.* [En línea] 2013. [Citado el: 6 de Febrero de 2016.] Lenguaje Unificado de Modelado. <http://www.scribd.com/doc/8963141/lenguaje-unificado-de-modelado>.

**Sánchez, Tamara Rodríguez. 2014.** Metodología de desarrollo para la Actividad Productiva UCI. Versión 1.2. Ciudad de la Habana : s.n., 2014.

**Sommerville, Ian. 2005.** Ingeniería del Software. Madrid : Pearson Education, 2005.

**Tracey, Karen. 2014.***LibrosWeb.* [En línea] 2014. [Citado el: 17 de Febrero de 2016.] [http://librosweb.es/libro/css\\_avanzado/capitulo\\_5.html](http://librosweb.es/libro/css_avanzado/capitulo_5.html).

**Villamarin, Alexis Sigcha. 2014.** Python IDE & Django IDE for Web developer: JetBrains PyCharm. *Jetbrains.com.* [En línea] 2014. [Citado el: 3 de Marzo de 2016.] <https://www.jetbrains.com/pycharm/>.

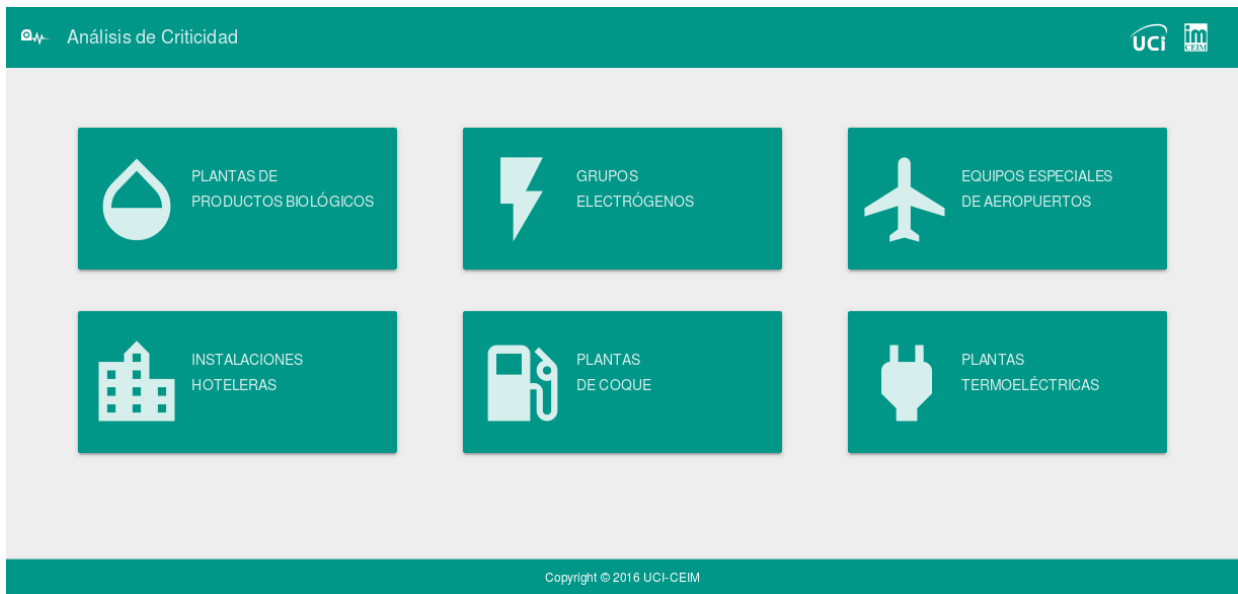
**Wiles, Frank. 2014.***Nebaris.* [En línea] 2014. [Citado el: 17 de Marzo de 2016.] <http://www.nebaris.com/post/126/que-es-bootstrap>.

**William, S. Davis. 1992.***Unirioja.es.* [En línea] 1992. [Citado el: 23 de Noviembre de 2015.] <http://dialnet.unirioja.es/servlet/libro?codigo=227309.8428319278>.

**Wilson, Jose Manuel Ayala. 2011.***Blogspot.* [En línea] 2011. [Citado el: 5 de Diciembre de 2015.] <http://jmaw.blogspot.com/2011/04/h2-margin-bottom-0.html>.

# ANEXOS

## Anexo 1:



*Figura 26: Vista de la pantalla principal del sistema 'Análisis de Criticidad'.*



## Anexo 2:

UCI | CIG-AF-CL09001 : Acta de liberación

### Control del documento

Título: Sistema de análisis de criticidad  
Versión: 1.0

	Nombre	Cargo
Elaborado por	Ing. Anabel Fé León Mendoza	RGA
Revisado por	Ing. Yisel Niño Benitez	Asesor de Calidad

Aprobado por	MsC. Maybel Díaz Capote	Firma
Cargo	Subdirector I+D+i	Fecha: 10/06/2016

### Reglas de confidencialidad

Clasificación: Uso Interno  
Forma de distribución: PDF Digital

Este documento contiene información propietaria del CENTRO DE INFORMATIZACIÓN DE ENTIDADES, y es emitido confidencialmente para un propósito específico.

El que recibe el documento asume la custodia y control, comprometiéndose a no reproducir, divulgar, difundir o de cualquier manera hacer de conocimientos público su contenido, excepto para cumplir el propósito para el cual se ha generado.

Las reglas son aplicables a las 7 páginas de este documento.

### Control de cambios

Versión	Lugar*	Tipo**	Fecha	Autor	Descripción
Versión 1.0	Todo el documento	A	10/06/2016	Ing. Anabel Fé León Mendoza	Creación

\* Sección del documento, Tabla, Figura  
\*\* A Alta, B Baja, M Modificación

CENTRO DE INFORMATIZACIÓN DE ENTIDADES  
Universidad de las Ciencias Informáticas  
Carretera a San Antonio Km 2 ½, Torrens,  
Boyeros, Ciudad de La Habana, Cuba  
Teléfono + 53 (7) 837 3680  
E-mail: software.gestion@uci.cu

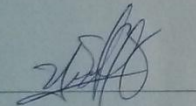
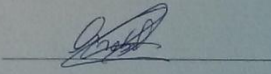
| 2

UCI | CIG-AF-CL09001 : Acta de liberación

### 4.2 Otro personal especializado participante:

Nombres y Apellidos	Cargo
Ing. Anabel Fé León Mendoza	RGA

 Ing. Yisel Niño Benitez Asesor de Calidad	 Ing. Anabel Fé León Mendoza RGA
---	--

**Figura 27:** Acta e Liberación del Producto.



### Anexo 3:

 UCI  
Universidad de las Ciencias  
Informáticas

"Año 58 del Triunfo de la Revolución"

### CARTA DE ACEPTACIÓN

Por medio de la presente se hace constar que los módulos de Criticidad incluidos en la Herramienta Informática para la Ingeniería de Mantenimiento cumplen con todos los requerimientos, logrando contribuir a mejorar los análisis de criticidad y posibilitando un aumento en la eficiencia del trabajo y toma de decisiones de los Ingenieros en Mantenimiento.

La Herramienta Informática cuenta con una interfaz amigable e intuitiva lo cual permite una fácil interacción con los usuarios, además, está dividido en 6 módulos completamente independientes en concordancia con los fines productivos de distintas instituciones, restringiendo el uso de un único módulo de acuerdo al contexto operacional de cada una de éstas, fortaleciendo así la integración y seguridad del sistema. Cuenta con varias funcionalidades que giran entorno a los análisis de criticidad, tales como: realizar cálculos de criticidad y complejidad (este último solo en casos específicos), realizar búsquedas, graficar valores de criticidad, histórico de los cálculos realizados, crear reportes, importar libros de cálculos, exportar gráficos en varios formatos.

Eusebio Gómez P  
Nombre y Apellidos del Autor/Tutor

Eusebio Gómez  
Nombre y Apellidos del Cliente

[Firma]  
Firma del Autor

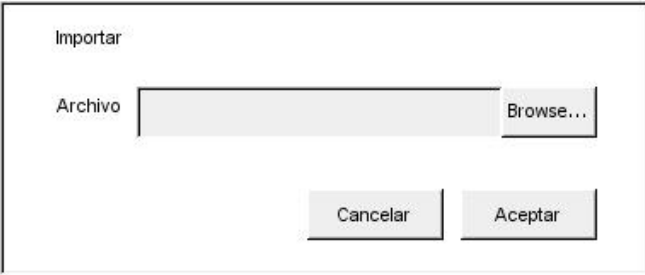
[Firma]  
Firma del Cliente

Fecha: 10, 06, 2016

**Figura 28:** Carta de Aceptación del Producto redactada por el Cliente.


#### Anexo 4:

*Tabla 9: Historia de Usuario del requisito funcional Importar Activos.*

Historia de Usuario	
<b>Número:</b> HU_1	<b>Nombre del requisito:</b> Importar activos
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 20 horas
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 9 horas
<b>Descripción:</b> Permitirá importar activos desde una hoja de cálculos previamente seleccionada por el usuario en la ubicación deseada.	
<b>Observaciones:</b> N/A	
<b>Prototipo de interfaz:</b>	
	

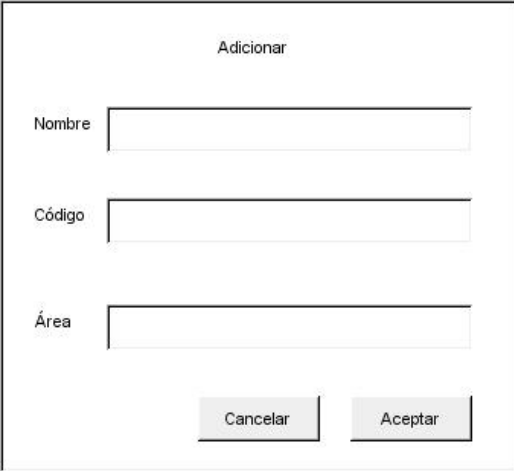
**Anexo 5:**

*Tabla 10: Historia de Usuario del requisito funcional Buscar Activo.*

<b>Historia de Usuario</b>	
<b>Número:</b> HU_3	<b>Nombre del requisito:</b> Buscar activo
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 18 horas
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 10 horas
<b>Descripción:</b> Permite realizar una búsqueda de un activo específico tanto por su nombre, como por su código, área o valor de criticidad dentro de toda la lista de activos.	
<b>Observaciones:</b> N/A	
<b>Prototipo de interfaz:</b>  	

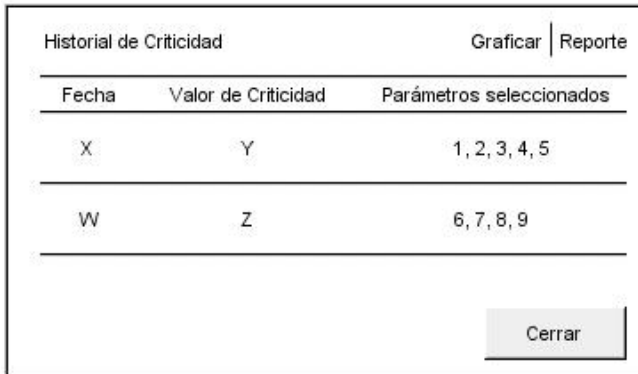
## Anexo 6:

*Tabla 10: Historia de Usuario del requisito funcional Adicionar Activo.*

Historia de Usuario	
<b>Número:</b> HU_7	<b>Nombre del requisito:</b> Adicionar Activo
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 24 horas
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 15 horas
<b>Descripción:</b> Permitirá adicionar al sistema un activo después de haber insertado cada uno de los parámetros necesarios para su inserción.	
<b>Observaciones:</b> N/A	
<b>Prototipo de interfaz:</b> 	


## Anexo 7:

*Tabla 10: Historia de Usuario del requisito funcional Mostrar Historial de Criticidad por Activo.*

Historia de Usuario	
<b>Número:</b> HU_14	<b>Nombre del requisito:</b> Mostrar historial de criticidad por activo
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 16 horas
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 8 horas
<b>Descripción:</b> Permite mostrar un historial con todos los valores de criticidad calculados previamente a un activo en específico, así como la fecha en que se realizaron dichos cálculos y los parámetros que fueron seleccionados en dicho cálculo.	
<b>Observaciones:</b> N/A	
<b>Prototipo de interfaz:</b>	
	

## Anexo 8:


*Tabla 10: Historia de Usuario del requisito funcional Mostrar Historial de Complejidad por Activo.*

Historia de Usuario																	
<b>Número:</b> HU_15	<b>Nombre del requisito:</b> Mostrar historial de complejidad por activo																
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1																
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 16 horas																
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 9 horas																
<b>Descripción:</b> Permite mostrar un historial con todos los valores de complejidad calculados previamente a un activo, así como la fecha en que se realizaron dichos cálculos y los parámetros que fueron seleccionados en dicho cálculo.																	
<b>Observaciones:</b> N/A																	
<b>Prototipo de interfaz:</b>																	
 <table border="1"><thead><tr><th colspan="2">Historial de Complejidad</th><th>Graficar</th><th>Reporte</th></tr><tr><th>Fecha</th><th>Valor de Complejidad</th><th colspan="2">Parámetros seleccionados</th></tr></thead><tbody><tr><td>X</td><td>Y</td><td colspan="2">1, 2, 3, 4, 5</td></tr><tr><td>W</td><td>Z</td><td colspan="2">6, 7, 8, 9</td></tr></tbody></table>		Historial de Complejidad		Graficar	Reporte	Fecha	Valor de Complejidad	Parámetros seleccionados		X	Y	1, 2, 3, 4, 5		W	Z	6, 7, 8, 9	
Historial de Complejidad		Graficar	Reporte														
Fecha	Valor de Complejidad	Parámetros seleccionados															
X	Y	1, 2, 3, 4, 5															
W	Z	6, 7, 8, 9															



### Anexo 9:

*Tabla 10: Historia de Usuario del requisito funcional Filtrar Reporte por Áreas.*

Historia de Usuario	
<b>Número:</b> HU_29	<b>Nombre del requisito:</b> Filtrar reporte por áreas
<b>Programador:</b> Yotsan Manuel Hernández Basulto	<b>Iteración Asignada:</b> 1
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 15 horas
<b>Riesgo en Desarrollo:</b> Poca experiencia por parte de los estudiantes con las tecnologías de desarrollo	<b>Tiempo Real:</b> 8 horas
<b>Descripción:</b> Permite filtrar por áreas el reporte de criticidad y/o complejidad en formato PDF, de manera tal que el mantenedor solo obtendrá en el PDF los datos de los activos que pertenezcan a las áreas seleccionadas, por defecto, de manera inicial el sistema filtra todas las áreas.	
<b>Observaciones:</b> N/A	
<b>Prototipo de interfaz:</b>	
	

## Anexo 10:

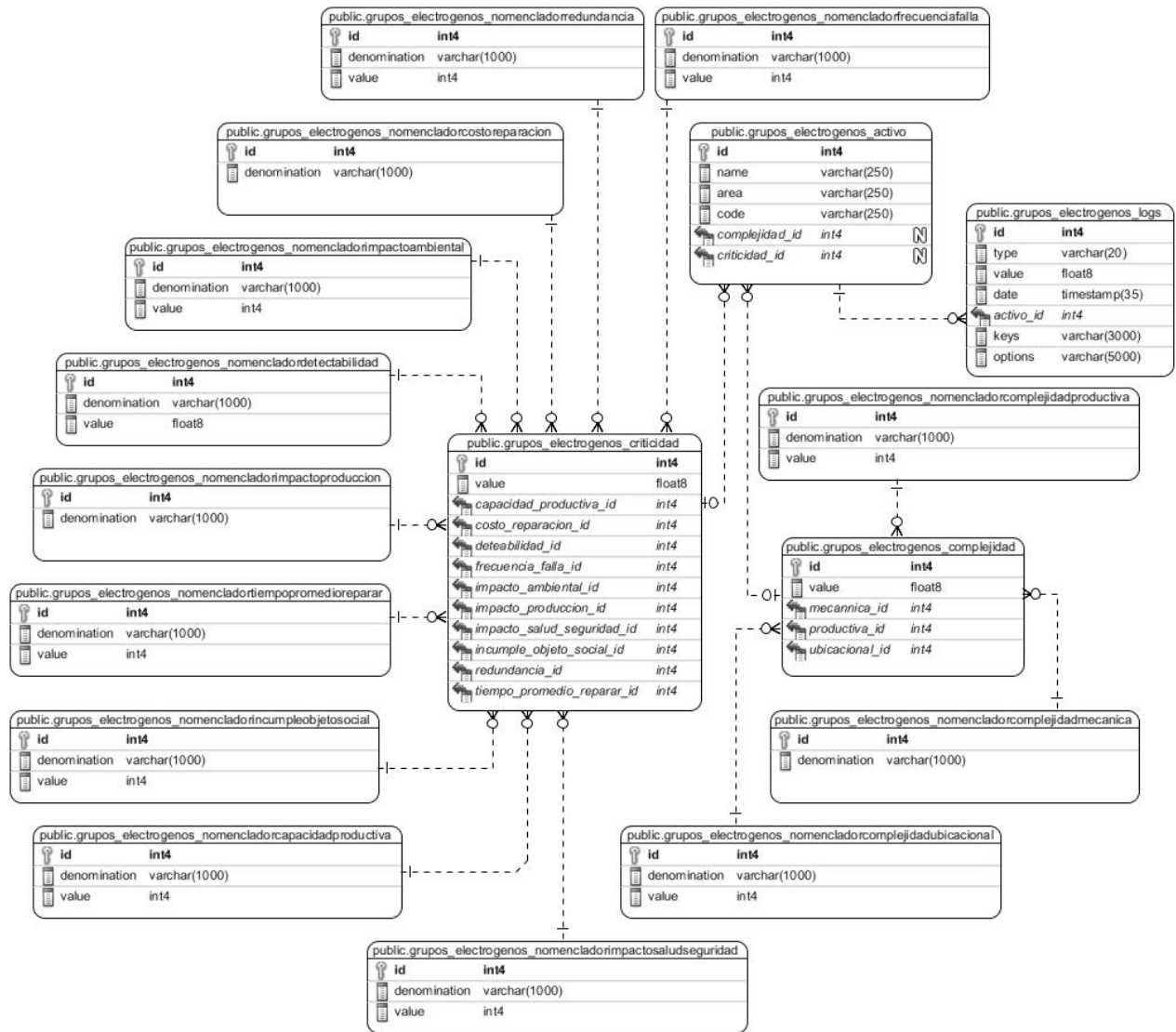
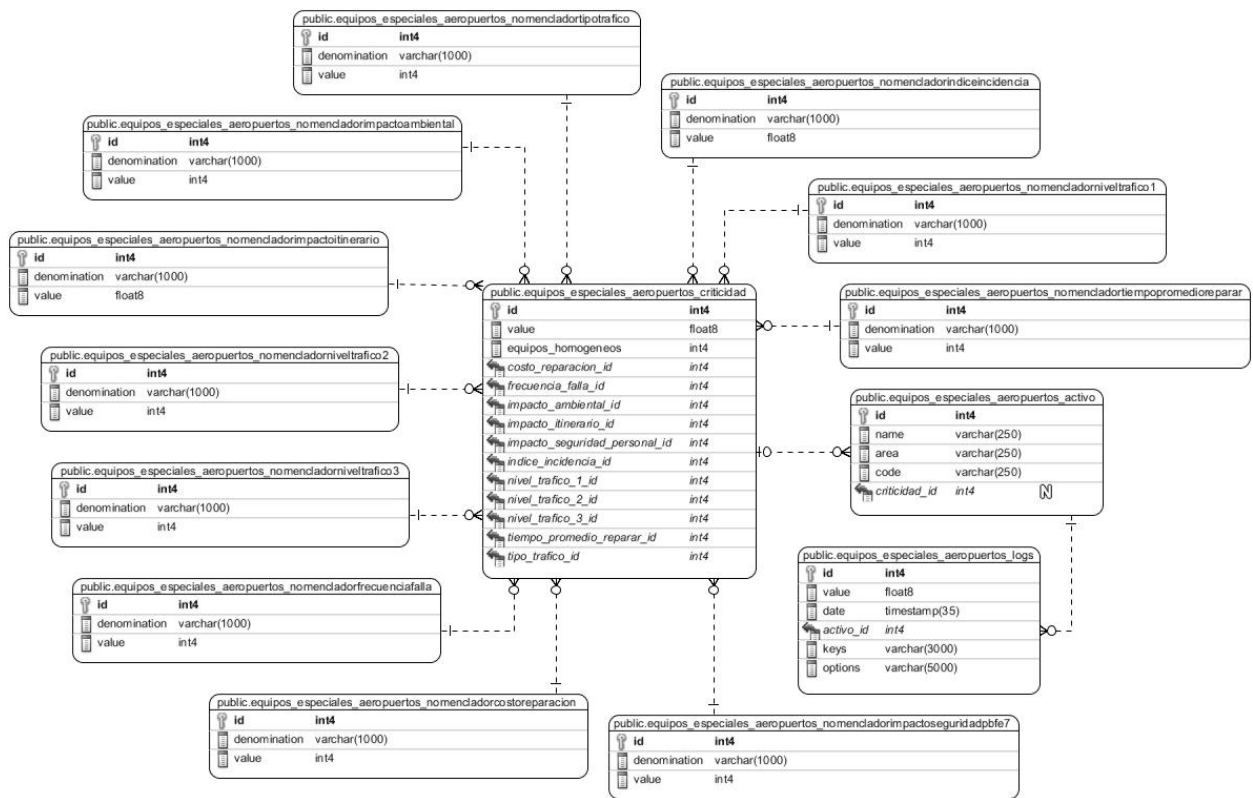


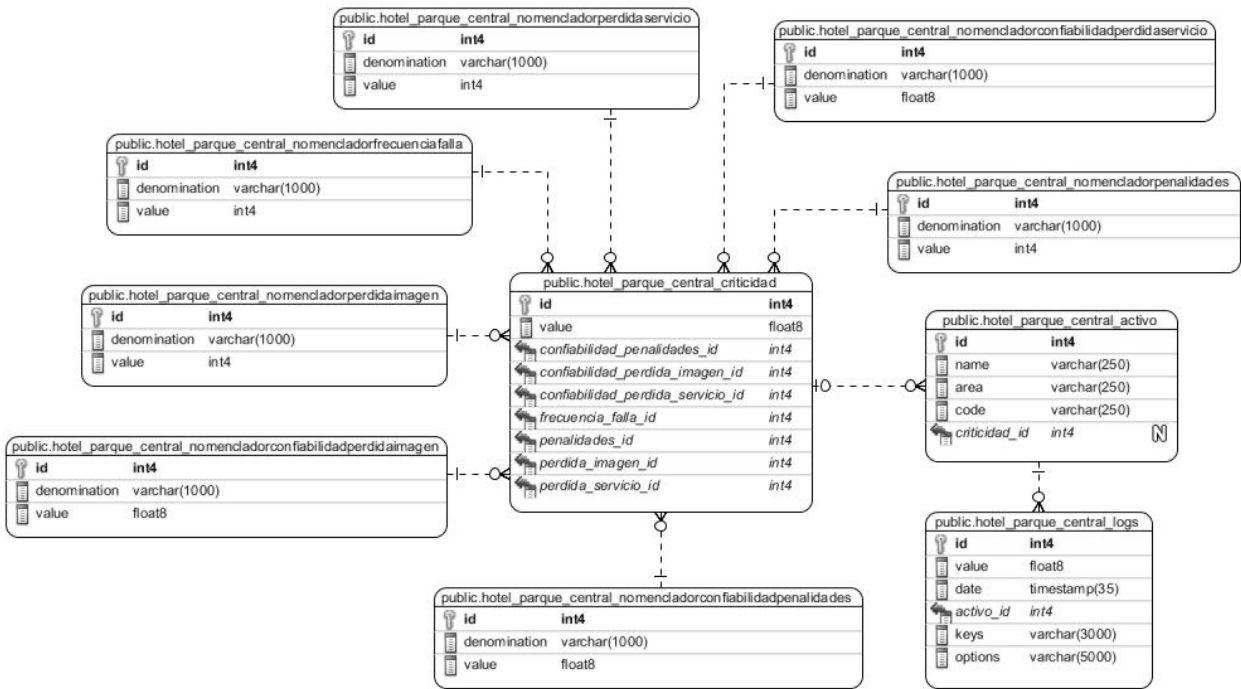
Figura 29: Modelo de Base de Datos del Módulo Grupos Eléctricos.

## Anexo 11:



**Figura 30:** Modelo de Base de Datos del Módulo Equipos Especiales de Aeropuertos.

## Anexo 12:



*Figura 31: Modelo de Base de Datos del Módulo Instalaciones Hoteleras.*

### Anexo 13:

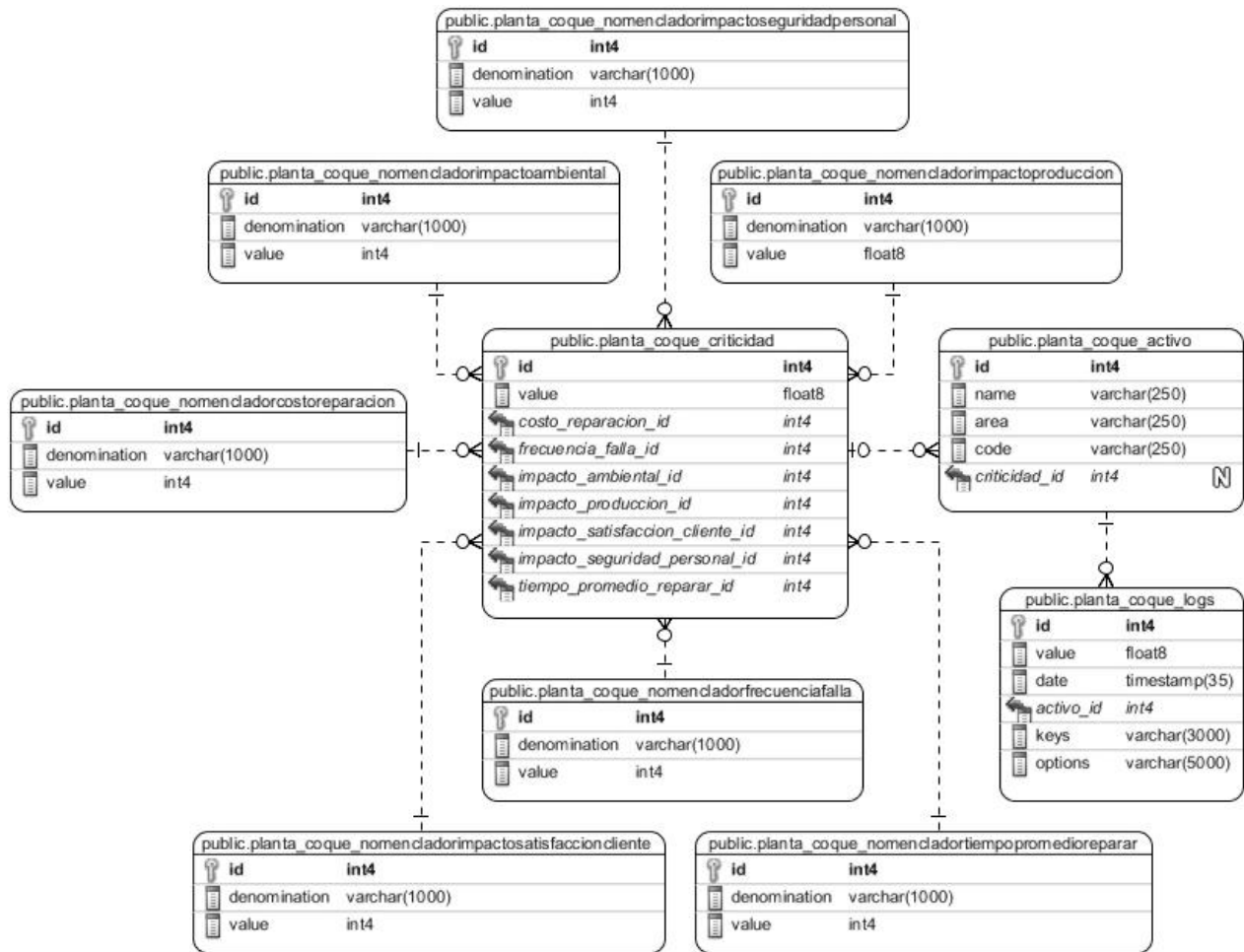


Figura 32: Modelo de Base de Datos del Módulo Plantas de Coque.

## Anexo 14:

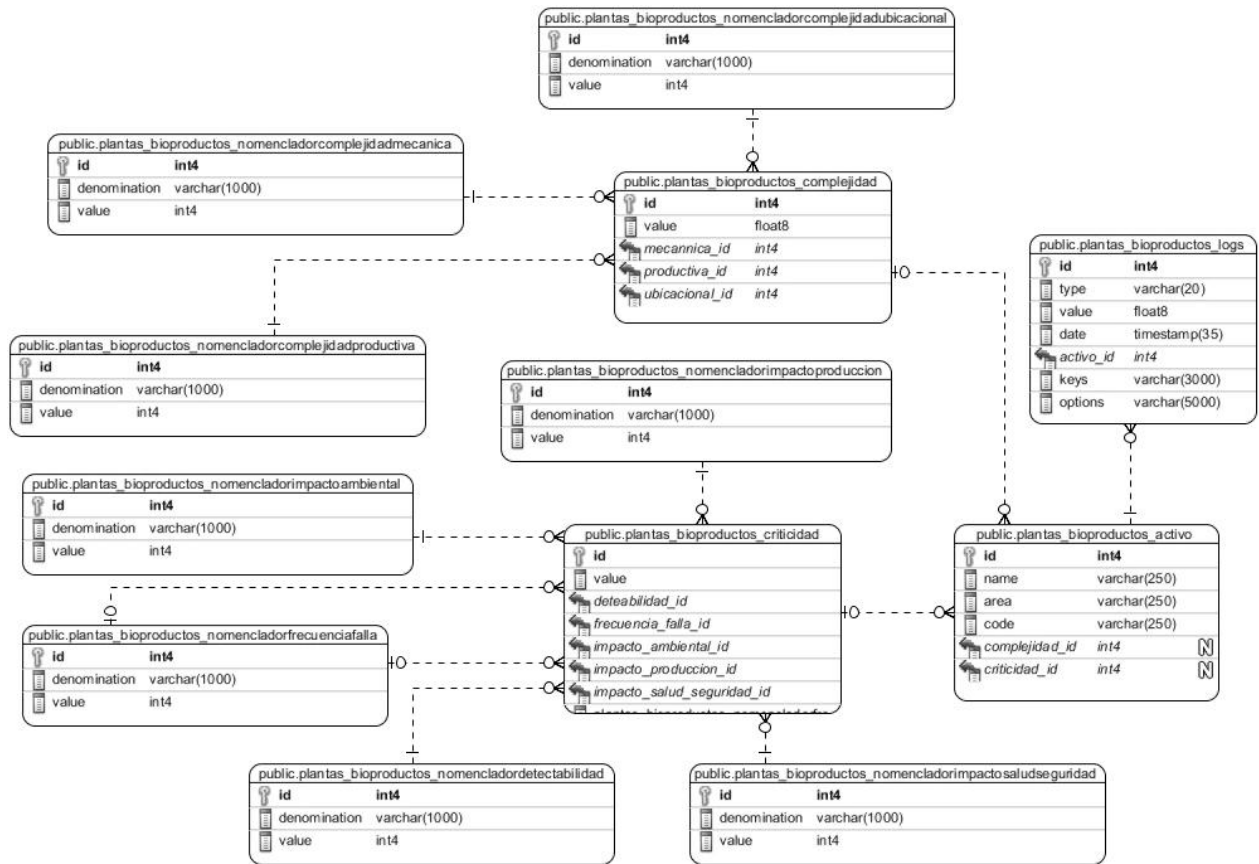


Figura 33: Modelo de Base de Datos del Módulo Plantas de Productos Biológicos.



## Anexo 16:

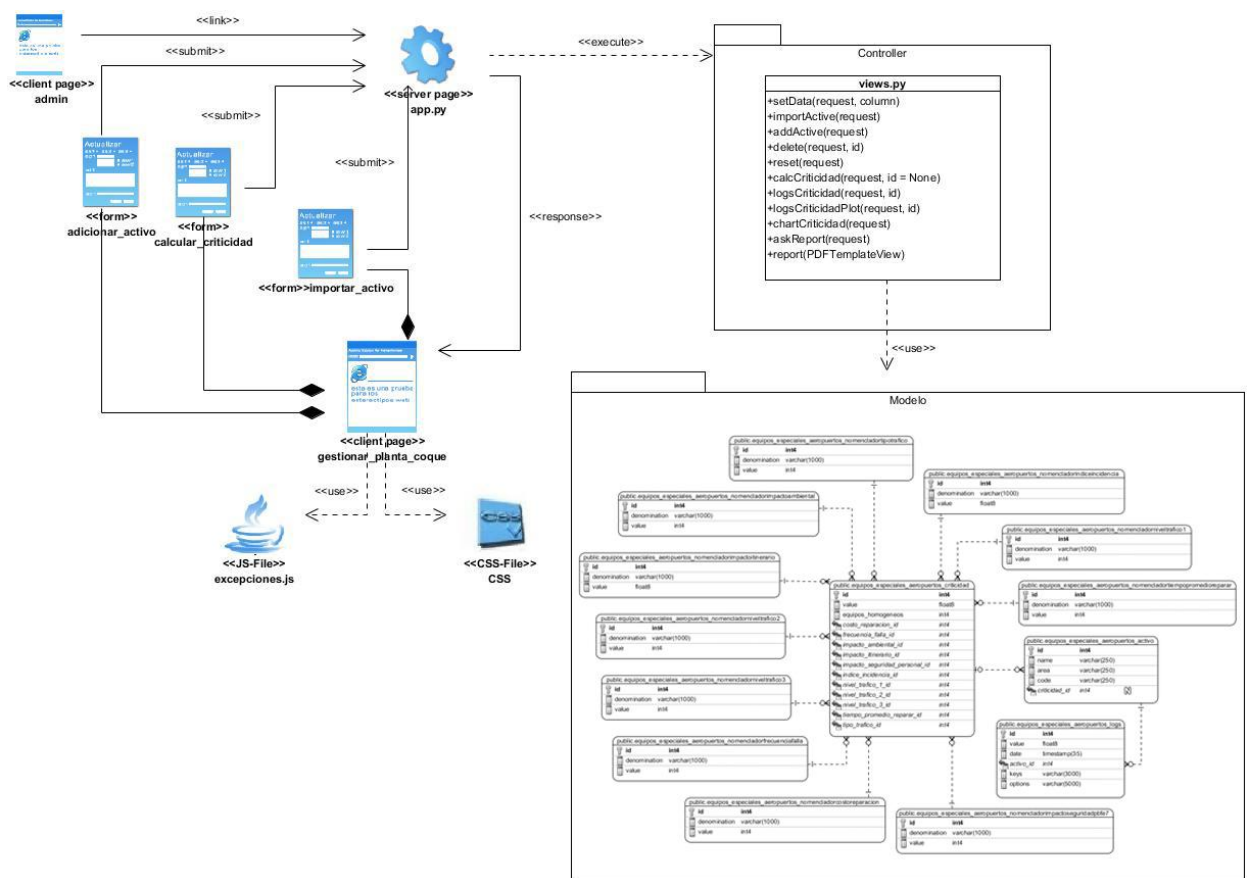


Figura 35: Diagramas de clases del diseño con estereotipos web: Gestionar Equipos Especiales de Aeropuertos.



## Anexo 17:

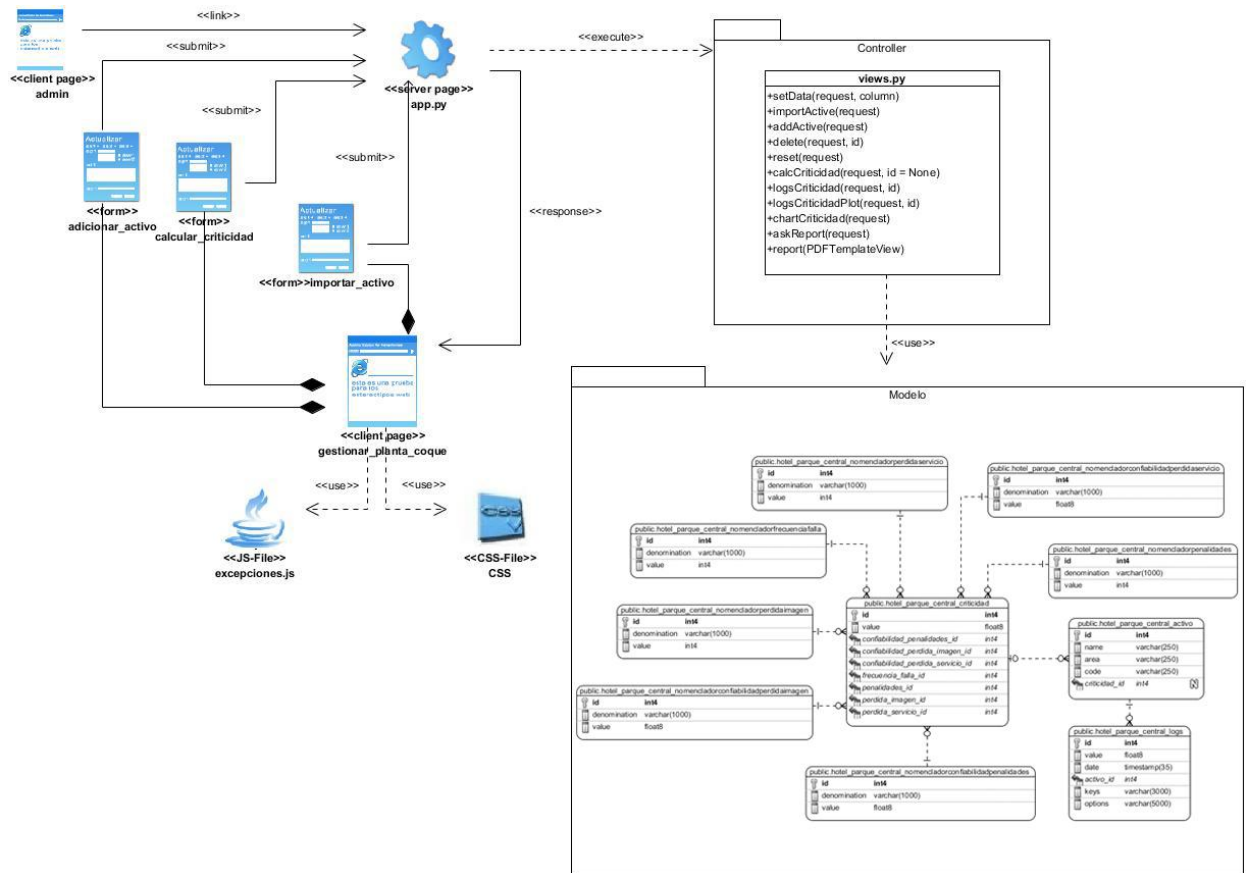


Figura 36: Diagramas de clases del diseño con estereotipos web: Gestionar Instalaciones Hoteleras.

## Anexo 18:

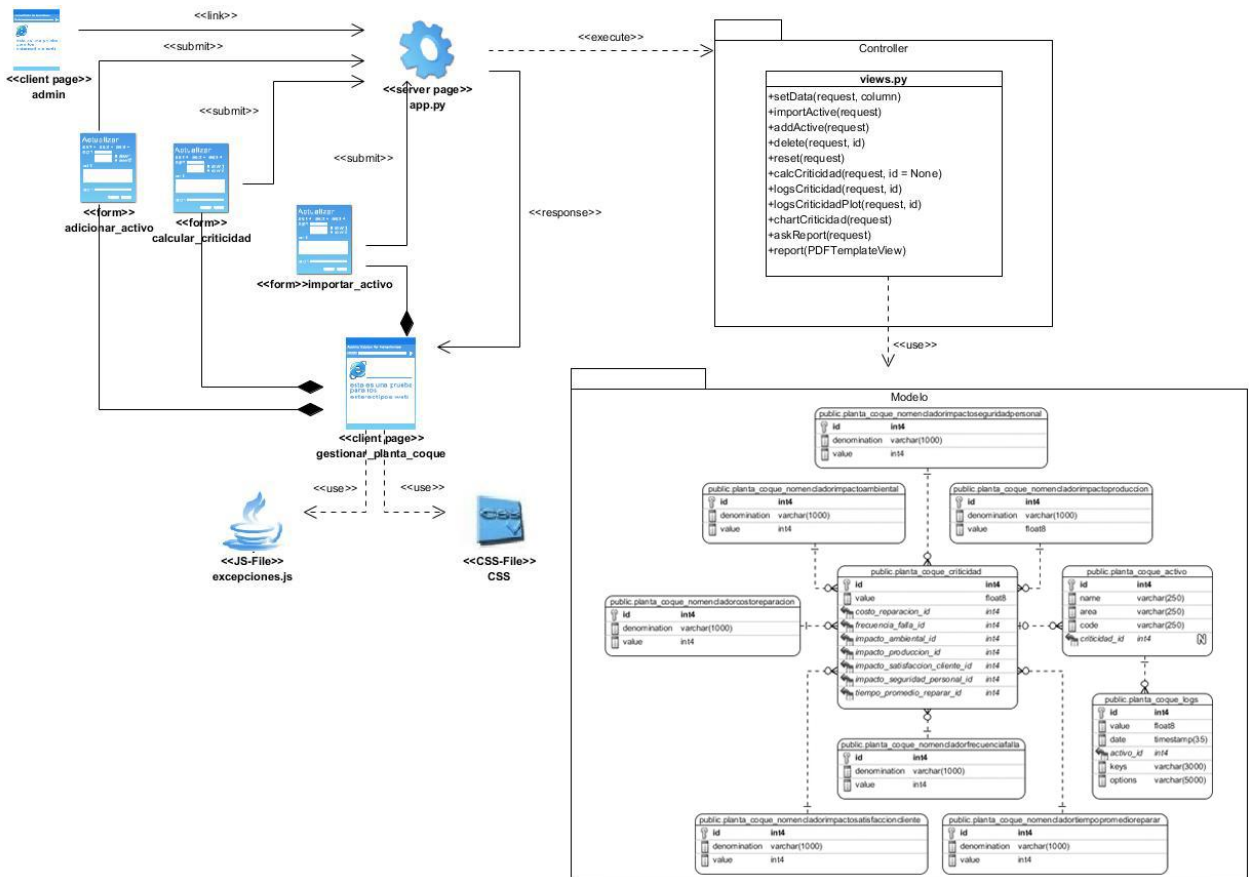


Figura 37: Diagramas de clases del diseño con estereotipos web: Gestionar Plantas de Coque.

## Anexo 19:

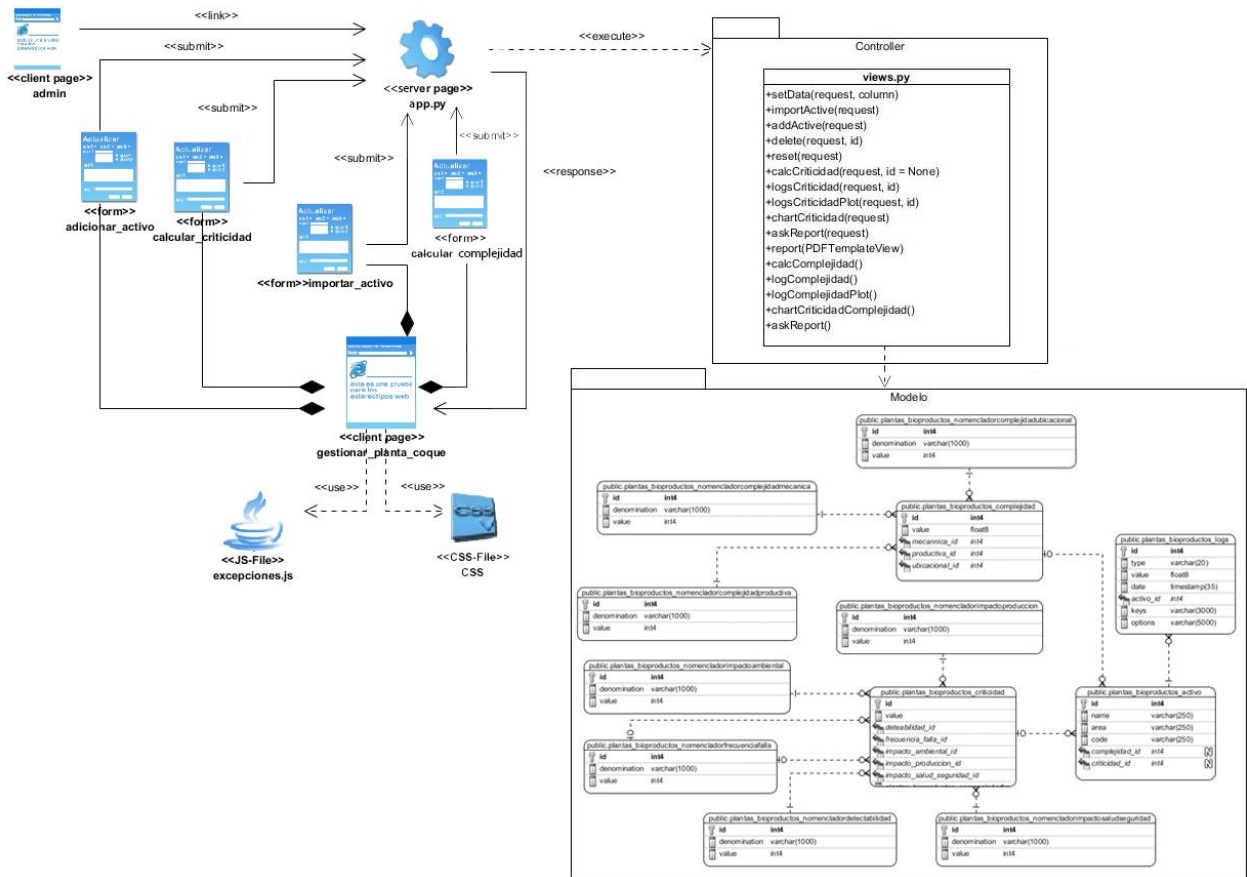


Figura 38: Diagramas de clases del diseño con estereotipos web: Gestionar Plantas de Productos Biológicos.