

Universidad de las Ciencias Informáticas

Facultad 3



Título: Desarrollo de un sistema para la gestión y análisis de las decisiones arquitectónicas en el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP.

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

Autores: Elizabeth Pérez Miranda


Lisvany Hernández Cabrera

Tutores: Ing. Daileny Caridad Arias Pupo

Ing. Giselle Almeida González

Ing. Karel Riverón Escobar

La Habana, junio de 2014. “Año 56 de la Revolución”.



“Se puede adquirir conocimientos y conciencia a lo largo de toda la vida, pero jamás en ninguna otra época de su existencia una persona volverá a tener la pureza y el desinterés con que, siendo joven, se enfrenta a la vida.”

Fidel Castro Ruz

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 _____.

Elizabeth Pérez Miranda

Firma del Autor

Lisvany Hernández Cabrera

Firma del Autor

Ing. Daileny Caridad Arias Pupo

Firma del Tutor

Ing. Giselle Almeida González

Firma del Tutor

Ing. Karel Riverón Escobar

Firma del Tutor

Datos de contacto

Síntesis del tutor:

Ing. Daileny Caridad Arias Pupo

Graduada de Ingeniera en Ciencias Informáticas en el año 2011. Especialista General de la Universidad de las Ciencias Informáticas (UCI), Jefa del proyecto Cedrux v1.1 con 5 años de experiencia en el desarrollo de software.

Correo electrónico: dcarias@uci.cu

Ing. Giselle Almeida González

Graduada de Ingeniera en Ciencias Informáticas en el año 2008. Especialista General con categoría docente Instructor, de la Universidad de las Ciencias Informáticas (UCI), Jefa del Laboratorio de Pruebas de Calidad del CEIGE con 6 años de experiencia productiva.

Correo electrónico: galmeida@uci.cu

Ing. Karel Riverón Escobar

Graduado de Ingeniero en Ciencias Informáticas en el año 2013. Recién Graduado en Adiestramiento. Es Asesor Tecnológico del CEIGE, tiene 3 años de experiencia en el desarrollo de software.

Correo electrónico: kre@uci.cu

Dedicatoria

Quisiera dedicarle esta tesis que evidencia el esfuerzo y resultado que he obtenido durante estos años de la carrera y que si no hubiera sido por ellos y por mi fuerza de voluntad de decir siempre ¡sí puedo! y voy a salir adelante, porque siempre he tenido la esperanza de que todos mis sueños se hagan realidad, a mis padres.

A mi madre querida que para mí hay pocas en este mundo por ser tan fuerte y luchadora, por darme todo su apoyo siempre que lo necesito y por ser mi mejor amiga en los buenos y malos momentos de la vida.

A mi papá por ser ese guía, ese compañero de estudios que siempre estuvo ahí apoyándome y guiándome.

A mi querida abuela que nos ayudó tanto cuando más lo necesitábamos.

A mi novio y futuro esposo Ubesnel Rojas Orozco, por darme la felicidad y por hacerme creer en el verdadero amor.

Y no por último es menos importante, en especial le dedico este resultado a mi hermano Manuel Alejandro Pérez Miranda, a ti mi hermanito del alma te dedico con todo el amor de una hermana mi trabajo de diploma.

De Elizabeth Pérez Miranda.

Agradecimientos

En especial quiero agradecer ser hoy la profesional que soy a mis padres, por haber sido capaces de traerme hoy hasta aquí, con todos los esfuerzos, preocupaciones y dolores de cabeza que eso ha implicado.

A mi mamá por ser una súper madre, por ser tan luchadora, incansable, por darme fuerzas cuando más lo necesito, por sus consejos, por hacer de mí una mujer de bien... Te quiero mucho mami...

A mi papá que ha estado siempre pendiente de mi superación, por ser mi guía en el estudio... Te quiero papá...

A mi querido hermanito del alma que por él he tenido las fuerzas de seguir siempre adelante y lograr ser hoy lo que soy... Te adoro...

A mi adorado novio y futuro esposo Ube por darme la felicidad que hoy siento, por amarme tanto, por ser de mí otra persona que pensó que no encontraría al verdadero amor, por ser todo lo que yo quería para mí... Te Amo...

A la familia de mi novio por acogerme con tanto cariño... los quiero...

A mis abuelos que hoy no están, pero donde quiera que estén siempre los tendré en mi corazón...

A todos mis tíos y primos por dar su granito de arena.

A José Eduardo Vázquez Hernández que me ayudó mucho en los primeros años de la carrera cuando ya lo veía todo perdido en la universidad, que si no hubiera sido por sus consejos y por los de mi madre hoy no estaría aquí parada...gracias de verdad...

A mis tutores Daileny, Giselle y Karel por su dedicación y apoyo durante el desarrollo de la tesis.

A mis compañeros de aula por compartir estos años de estudio con ellos, en especial a Raciél Roche Escobar por sus horas y dedicación incondicional de estudio siempre que lo necesitaba.

A mis compañeros y profesionales del proyecto...

A la UCI, porque me ha permitido crecer, como profesional y como persona, gracias por todas las oportunidades que me ha brindado.

A todos los que permitieron que hoy estuviera escribiendo estas letras muchas gracias...

De Elizabeth Pérez Miranda.

Resumen

El uso de aplicaciones informáticas para dar solución a diferentes problemas y situaciones de la vida cotidiana constituyen una práctica común por estos días. Construir aplicaciones capaces de adecuarse a los distintos escenarios donde sean aplicadas de manera rápida y confiable se ha convertido en una necesidad. En la Universidad de las Ciencias Informáticas se desarrolla el Sistema Integral de Gestión Xedro-ERP, destinado a la gestión económica de las entidades del país. Dentro del desarrollo de este software, se presentan una serie de problemas arquitectónicos que hacen que el trabajo se haga más engorroso. Para ello se toman decisiones, que son a menudo esenciales para el correcto desempeño del mismo.

En el presente trabajo de diploma se realiza una propuesta de solución, que permite gestionar las decisiones arquitectónicas en el proceso de desarrollo de Xedro-ERP, de modo que posibilite la identificación de posibles patrones en el entorno arquitectónico en el que se desenvuelvan. Para ello se llevó a cabo el modelo de dominio, el levantamiento de requisitos, análisis y diseño, implementación y pruebas de caja blanca, caja negra y de aceptación; además se emplearon varias herramientas y tecnologías que garantizaron la implementación del sistema.

PALABRAS CLAVE

Arquitectura de software, decisiones arquitectónicas, patrones.

Índice de contenidos

Introducción	1
Capítulo 1: Fundamentos Teóricos	5
1.1 Introducción	5
1.2 Conceptos básicos relacionados con el dominio del problema	5
1.3 Estado del arte.....	7
1.4 Técnicas de Minería de texto	9
1.5 Modelo de desarrollo de software	9
1.6 Ambiente de desarrollo	11
1.7 Patrones de diseño	15
1.8 Conclusiones del capítulo	16
Capítulo 2: Análisis y Diseño de la Solución Propuesta	17
2.1 Introducción	17
2.2 Selección del experto humano.....	17
2.3 Análisis de la encuesta aplicada	18
2.4 Técnicas para la Captura de requisitos	19
2.5 Requisitos de software.....	19
2.6 Técnicas para la Validación de Requisitos	23
2.7 Modelo de Diseño	23
2.8 Modelo de datos	25
2.9 Patrones de diseño utilizados	26
2.10 Validación del diseño mediante métricas	27
2.11 Conclusiones del capítulo	36
Capítulo 3: Implementación y Prueba	38
3.1 Introducción	38
3.2 Descripción de la arquitectura.....	38
3.3 Diagrama de componentes	39
3.4 Aspectos importantes de la implementación del sistema	40
3.5 Diagrama de despliegue	42
3.6 Pruebas de software	42
3.7 Validación de las variables de investigación	49
3.8 Conclusiones del capítulo	49
Conclusiones Generales	50
Recomendaciones	51
Bibliografía	52
Anexos	55

Glosario.....	70
---------------	----

Índice de figuras

Figura 1: Gestionar decisión.....	24
Figura 2: Adicionar decisión.....	25
Figura 3: Modelo de datos.....	26
Figura 4: Representación de la evaluación de la métrica TOC.....	30
Figura 5: Representación en por ciento de los resultados obtenidos en la evaluación de la métrica TOC.	31
Figura 6: Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.	32
Figura 7: Resultados de la evaluación de la métrica TOC para el atributo complejidad de implementación..	32
Figura 8: Resultados de la evaluación de la métrica TOC para el atributo reutilización.	33
Figura 9: Representación en por ciento de los resultados obtenidos en los intervalos definidos según la métrica RC.....	34
Figura 10: Resultados de la evaluación de la métrica RC para el atributo acoplamiento.....	34
Figura 11: Resultados de la evaluación de la métrica RC para el atributo complejidad de mantenimiento....	35
Figura 12: Resultados de la evaluación de la métrica RC para el atributo reutilización.....	35
Figura 13: Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas.....	36
Figura 14: Diseño arquitectónico.	39
Figura 15: Diagrama de componentes.....	40
Figura 16: Matriz de situaciones por términos.....	41
Figura 17: Representación en matrices del algoritmo Descomposición de Valores Singulares.....	41
Figura 18: Diagrama de despliegue.....	42
Figura 19: Código fuente de la funcionalidad semejanzaAction(Request \$request).....	45
Figura 20: Grafo de flujo asociado a la funcionalidad semejanzaAction(Request \$request).....	45
Figura 21: Por ciento que representan las No conformidades detectadas en la documentación para cada una de las iteraciones.....	48
Figura 22: Por ciento que representan las No conformidades detectadas en la aplicación por cada una de las iteraciones.....	48

Índice de tablas

Tabla 1: Métrica Tamaño Operacional de Clase	28
Tabla 2: Rango de valores para la métrica TOC	28
Tabla 3: Atributos de calidad evaluados por la métrica RC	29
Tabla 4: Rango de valores para la métrica RC.....	29
Tabla 5: Tabla de las no conformidades detectadas en la documentación y en la aplicación por iteraciones.....	47

Introducción

En las últimas décadas del siglo XX junto con los avances de las Tecnologías de la información y las comunicaciones ha venido evolucionando el desarrollo del software. La Arquitectura de Software (AS) no está alejada a esta realidad, esta constituye un punto clave en el proceso de desarrollo del software, por lo que además de buscar un diseño sólido es preciso valorar las decisiones tomadas en esta disciplina. También brinda la posibilidad de evaluar el grado en que se han alcanzado los atributos de calidad que se persiguen y pone al descubierto tanto los puntos débiles como las potencialidades de la arquitectura propuesta. En la vida cotidiana se lleva a cabo la toma de decisiones (TD) para resolver distintas situaciones que se presentan; esta consiste en elegir una opción entre las disponibles, a los efectos de resolver un problema actual o potencial, aún cuando no se evidencie un conflicto latente. La calidad de las decisiones tomadas marca la diferencia entre el éxito o el fracaso. En toda empresa se hace necesaria la TD que son a menudo esenciales para el correcto desempeño de la misma; muchas veces estas son tomadas gracias al servicio de un software diseñado para este fin (1).

Cuba se encuentra inmersa en el desarrollo de aplicaciones informáticas que posibiliten informatizar la sociedad, muestra de ello lo constituye la Universidad de las Ciencias Informáticas (UCI). Este centro de altos estudios, surge en medio del deseo de crecer en el desarrollo tecnológico, e implementar sistemas tanto para beneficio propio como para el país. La Universidad está compuesta por 7 facultades y a la vez cada una de ellas está integrada por varios centros productivos, entre ellos se encuentra el Centro de Informatización de la Gestión de Entidades (CEIGE) perteneciente a la Facultad 3. En el mismo se desarrolla el Sistema Integral de Gestión Xedro-ERP para la planificación de los recursos empresariales. Durante el desarrollo del mismo el equipo de arquitectura se enfrentó a una serie de dificultades enunciadas a continuación:

- En la comunicación entre los diferentes subsistemas, módulos y componentes.
- En el trabajo de campos dinámicos en la base de datos.
- Con el rendimiento del sistema debido a los grandes volúmenes de datos y los extensos procesos presentes en el negocio.

Esta situación generó un conjunto de soluciones que comenzaron a diseñarse e implementarse por cada uno de los equipos del proyecto, pero sin homogeneidad, sin una adecuada documentación, sin una adecuada gestión de la relación solución – problema, sin una adecuada definición de la taxonomía¹ de los tipos de problemas identificados. Estas malas prácticas generan una serie de dificultades, entre las que se destacan:

- No existe una adecuada documentación de las soluciones dadas, de manera que permita su entendimiento y posible reutilización en otros desarrollos.
- No existe una adecuada gestión de los problemas o dificultades arquitectónicamente significativas, siendo el resto de las soluciones vulnerables a problemas resueltos en otras partes del sistema.
- No se incorpora a la documentación técnica de la arquitectura las soluciones dadas, por no tener una clasificación de las mismas, atendiendo a las vistas arquitectónicas definidas.
- Dificulta la identificación de posibles patrones de la arquitectura dentro del entorno de desarrollo, imposibilitando la reutilización de soluciones que ya se hayan identificado anteriormente.

Estas deficiencias aumentan los riesgos a fallas en la arquitectura del sistema, imposibilitando la reutilización de buenas prácticas y patrones propios dentro del sistema, afectando la calidad y el impacto del mismo.

Partiendo de lo antes expuesto se plantea como **problema a resolver**:

La forma en que se realiza la documentación de las decisiones arquitectónicas tomadas durante el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP, dificulta la gestión de las decisiones arquitectónicas así como la identificación de posibles patrones en el entorno arquitectónico del desarrollo del sistema.

Para darle respuesta a esta problemática se define como **objeto de estudio**:

Arquitectura de software, centrado en el **campo de acción** Decisiones arquitectónicas.

Basado en el problema planteado se determina como **objetivo general**:

Desarrollar un sistema que permita gestionar las decisiones arquitectónicas en el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP, de modo que posibilite la identificación de posibles patrones en el entorno arquitectónico en el que se desenvuelvan.

Para lograr el cumplimiento del objetivo general se trazaron los siguientes **objetivos específicos**:

¹ **taxonomía**: "Ciencia que estudia los principios, métodos y fines de la clasificación de los seres vivos; clasificación u ordenación en grupos de cosas que tienen unas características comunes" (12).

1. Establecer el marco teórico de la investigación para fundamentar conceptos, métodos y herramientas.
2. Realizar la especificación y descripción de requisitos del sistema para identificar las posibles funcionalidades.
3. Diseñar la solución de software para facilitar la implementación del sistema.
4. Implementar el diseño de la solución de software para favorecer la gestión de las decisiones arquitectónicas que se tomen durante el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP.
5. Validar la solución de software propuesta mediante la aplicación de pruebas de Caja Blanca, Caja Negra y de Aceptación.

Se propone como **idea a defender**:

El desarrollo de un sistema para la gestión de las decisiones arquitectónicas en el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP, permitirá la identificación de posibles patrones en el entorno arquitectónico en el que se desenvuelvan.

Los siguientes **métodos teóricos** sustentan el trabajo de investigación:

- **Histórico – Lógico:**

Este método permitió realizar un estudio de las principales soluciones relacionadas con la investigación a nivel mundial. De esta manera se pudo conocer acerca de la existencia y características de la herramienta analizada y los modelos vinculados con el tema de investigación.

- **Analítico-Sintético:**

Con el objetivo de analizar teorías y documentos, y a partir de ello realizar un estudio previo acerca de las características que aportaron los modelos estudiados, se fortaleció la investigación con los principales elementos relacionados con el objeto de estudio y la necesidad del sistema que se propone.

- **Modelación**

Con el objetivo de realizar abstracciones para dar una explicación de la realidad existente. Su condición principal es la relación entre el modelo y el objeto que se modela, que en este caso sería el sistema en cuestión. Se evidencia el uso de este método en el Modelo conceptual realizado.

El siguiente **método Empírico** sustenta el trabajo de investigación:

- **Encuesta:**

Este método permitió la obtención de información relacionada con la investigación a través de una encuesta pre elaborada a arquitectos del CEIGE y expertos en el tema de decisiones arquitectónicas para conocer las dificultades que presentan a la hora de tomar decisiones.

La presente investigación está estructurada en tres capítulos:

➤ **Capítulo 1 Fundamentación teórica:**

Se hace referencia a los principales conceptos claves para la investigación. Se realiza un análisis del estado del arte. Se fundamentan las herramientas, metodologías y tecnologías para facilitar el desarrollo de la solución propuesta.

➤ **Capítulo 2 Análisis y Diseño:**

En este capítulo se exponen los resultados arrojados de la encuesta aplicada a los arquitectos y especialistas en el tema. También los requisitos funcionales y no funcionales identificados, así como los patrones de diseño utilizados. Se muestran los diferentes artefactos generados durante la disciplina de Análisis y diseño.

➤ **Capítulo 3 Implementación y Prueba:**

Este capítulo se enfoca en la construcción de la solución explicando los aspectos principales de la implementación. Se muestran los resultados de validar el diseño mediante las métricas Tamaño Operacional de Clase y Relaciones entre Clases. Se realiza la aplicación de pruebas de Caja blanca, Caja negra y de Aceptación para el correcto funcionamiento del sistema.

Capítulo 1: Fundamentos Teóricos

1.1 Introducción

En este capítulo se abordan los aspectos fundamentales que sirven de soporte teórico para el desarrollo de toda la investigación. Además se analizan los diferentes sistemas existentes en el ámbito nacional e internacional que gestionen los procesos correspondientes a la investigación. Por último, se realiza una breve caracterización de las tecnologías, metodologías y herramientas utilizadas para la solución del problema planteado.

1.2 Conceptos básicos relacionados con el dominio del problema

Para facilitar la comprensión de dicha investigación se deben entender primeramente los términos que se emplean durante el desarrollo de la misma. Se especifican algunos conceptos que no son comunes en el lenguaje cotidiano.

1.2.1 Arquitectura de software

El término de arquitectura de software, nace a partir de los sistemas de mediana y gran envergadura que proponen una solución a un problema específico. Dentro de las definiciones existentes se pueden citar algunas que son muy difundidas en todo el mundo.

Una de ellas expuestas en el Libro Ingeniería del Software, un enfoque práctico, la cual plantea que: *“La arquitectura de software de un programa o computación es la estructura del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos”* (13).

- **Según Clements:**

“La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones” (2).

- **Según Kazman (1996):**

“La estructura de un sistema, la cual abarca componentes de software, propiedades externas visibles de estos componentes y sus relaciones” (3).

- La definición de AS que brinda el documento de **IEEE 1471-2000**, adoptada también por Microsoft:

“La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución” (3).

Al hacer un análisis detallado de cada uno de los conceptos disponibles de AS, resulta interesante la existencia de ideas comunes entre los autores dado que coinciden en que una AS define la estructura del sistema. La intención primordial del análisis no es concluir ni proponer un concepto que englobe todas las ideas planteadas hasta el momento, sino establecer aquellos elementos que no deben perderse de vista al momento de introducirse en el contexto de las AS, y por ende, en un ambiente de evaluación de AS. Por esta razón antes expuesta y pensada por expertos en el tema se asume que la definición más completa sea la del documento **IEEE 1471-2000**.

1.2.2 Toma de decisiones

“Proceso que consiste en realizar una elección entre diversas alternativas. La TD puede aparecer en cualquier contexto de la vida cotidiana, ya sea a nivel profesional, sentimental, familiar, etc. El proceso, en esencia, permite resolver los distintos desafíos a los que se debe enfrentar una persona o una organización” (4).

1.2.3 Decisiones arquitectónicas

“Las decisiones arquitectónicas son las decisiones de diseño conscientes que pueden afectar al sistema de software como un todo, o uno o más componentes de la base. Las decisiones arquitectónicas son de gran importancia porque pueden determinar directa o indirectamente si un sistema cumple con los requisitos no funcionales, como los atributos de calidad del software” (5).

“Las decisiones arquitectónicas son una de las formas más significativas de conocimiento arquitectónico. La mayoría de las decisiones-arquitectónicas no están documentadas y no se pueden derivar explícitamente de los modelos arquitectónicos. Simplemente existen en la forma de conocimiento contenido en las cabezas de los arquitectos o los otros interesados, e inevitablemente se disipan” (6).

Analizando los conceptos anteriores se llega a la conclusión de que las decisiones arquitectónicas son las soluciones que se le dan a los problemas que pueden existir en la arquitectura de un sistema.

1.2.4 Patrón

“Se conoce como patrón a un cúmulo de información que proporciona respuesta a un conjunto de problemas similares en un contexto dado. Los patrones hacen más resistente al cambio la producción de software y establecen parejas problema-solución. También ayudan a especificar interfaces, facilitan la reutilización del código y permiten una fácil comprensión debido a la documentación estándar que estos presentan” (1).

1.3 Estado del arte

Durante la investigación realizada sobre las decisiones arquitectónicas en el proceso de desarrollo de software actual, se encontraron algunos trabajos investigativos relacionados con la gestión de las decisiones arquitectónicas en el proceso de desarrollo de software.

ADvISE (Architectural Design Decision Support Framework):

ADvISE es un marco de trabajo de apoyo a las decisiones arquitectónicas de diseño. La misma contiene la definición de modelos de decisiones arquitectónicas basados en preguntas, opciones y criterios. Además permite crear, modificar y persistir modelos de decisiones arquitectónicas. Esta tiene como desventaja que a pesar de ser una herramienta de código abierto no permite gestionar problemas arquitectónicos ni reutilizar decisiones tomadas con anterioridad (10).

Modelos sobre las Decisiones arquitectónicas

Haciendo un estudio de lo relacionado con las decisiones arquitectónicas uno de los modelos encontrados ofrece una comparación de las técnicas de decisiones arquitectónicas existentes, escrito por varios autores, teniendo como tema: Técnicas de TD para la Arquitectura de Software, en el mismo plantean que las decisiones arquitectónicas son finalmente cruciales para el éxito de un proyecto de software, con el objetivo de guiar a los arquitectos en su selección. También se propone:

- Un esquema de caracterización que es capaz de diferenciar entre las diferentes técnicas de TD y revela claramente la variabilidad y puntos comunes entre ellos.
- Una validación de ese esquema, al usarla para caracterizar 15 técnicas de TD existentes.
- Una tabla de clasificación de las técnicas de TD que apoya su selección por los arquitectos.

La mayor parte del contenido del artículo proviene de análisis de la literatura sobre la arquitectura de software y técnicas de TD. Según los resultados de la investigación revelan que la selección de una técnica de TD es difícil, ya que no es posible afirmar que cualquier técnica es inequívocamente mejor que otra. En otras palabras, no existe una técnica de TD perfecta que domine todas las otras. A modo de resumen, se describen estas técnicas como alternativas a disposición de los arquitectos para elegir entre un conjunto finito de alternativas arquitectónicas (11).

Modelos estudiados

- Identificación de Decisiones arquitectónicas en los patrones arquitectónicos (modelo de decisión arquitectónica reusable) (27).
- Modelos de Decisión arquitectónica como Micro-Metodología para el Análisis Orientada a Servicios y Diseño (Arquitectura Orientada a Servicios (SOA)) (28).
- Modelos de Decisión arquitectónica reutilizables para el Desarrollo de Aplicaciones Empresariales (29).
- Software reutilizado para decisiones arquitectónicas en la Industria Eléctrica (30).
- Equipo de conciencia de la situación y TD de arquitectura con el almacén de Arquitectura de Software (31).

Valoración de los sistemas vinculados a la investigación

El estudio de trabajos investigativos van dirigidos a facilitar la gestión de las decisiones arquitectónicas en el proceso de desarrollo de software; estos sistemas tiene puntos comunes con la aplicación que se desea desarrollar. Ejemplo de ello:

- Contiene las decisiones arquitectónicas - puntos de decisión, y para cada punto de decisión, se introduce una serie de preguntas, junto con las posibles opciones o respuestas.
- La selección de una opción puede conducir bien a una solución (a menudo el patrón de base); una pregunta que requiere una respuesta en texto libre también puede ser seguida por una próxima decisión o pregunta.
- Tienen como objetivo apoyar a los arquitectos de software en la toma de decisiones de la arquitectura.

Estos sistemas aunque tienen puntos comunes con la aplicación que se quiere desarrollar, gestionan las decisiones arquitectónicas en correspondencia con sus características.

Además son sistemas personalizados para el ambiente en el que operan. Luego de analizar las particularidades de los modelos estudiados se llega a la conclusión de que son insuficientes de forma individual para responder a la problemática planteada. Dichos modelos solo permiten la documentación del problema, y más allá de documentar se necesita, partiendo de un problema, asociar las decisiones arquitectónicas que puedan ser aplicadas para cada caso.

1.4 Técnicas de Minería de texto

Para el desarrollo de la solución fue necesario el análisis de minería de textos, debido a la importancia de su uso en la investigación. Seguidamente se explica en qué consiste la misma y qué técnicas se utilizaron.

La Minería de texto, también conocida como minería de información en documentos (document information mining), minería de datos texto, o descubrimiento de conocimiento en bases de datos textuales es una fusión de tecnologías para analizar colecciones grandes de documentos sin estructura, con el propósito de extraer modelos (patrones) interesantes y no triviales o conocimiento. El objetivo es extraer la información hasta el momento no descubierta de colecciones grandes de texto. También se ha definido como: *“La extracción no trivial de información implícita, previamente desconocida y potencialmente útil de grandes cantidades de datos texto”* (32).

Basándonos en el estudio realizado para el proceso de minería de texto en la implementación del sistema se decide utilizar la técnica análisis semántico latente (LSA). Es una técnica en el procesamiento del lenguaje natural, en particular en la semántica vectorial, que consiste en analizar las relaciones entre un conjunto de documentos y los términos que contienen al producir una serie de conceptos relacionados con los documentos y los términos. LSA asume que las palabras que están cerca en sentido ocurrirán en piezas similares de texto. Una matriz que contiene recuentos de palabras por párrafo (filas y columnas representan las palabras únicas de cada párrafo) se construye a partir de un pedazo grande de texto y una técnica matemática llamada descomposición de valor singular (SVD) se utiliza para reducir el número de columnas mientras que preserva la estructura de similitud entre filas. Las palabras se comparan entonces tomando el coseno del ángulo entre los dos vectores formados por dos filas. Los valores cercanos a 1 representan palabras muy similares, mientras que valores cercanos a 0 representan palabras muy diferentes (33).

1.5 Modelo de desarrollo de software

Para lograr el desarrollo de la solución se hace uso del modelo de desarrollo que propone el CEIGE; el cual lleva como nombre Modelo de Desarrollo de Software CEIGE. El mismo proporciona una guía para regir el proceso de desarrollo de software centrado en la arquitectura y en el desarrollo de componentes como base tecnológica.

Este modelo posee las siguientes características:

Orientado a componentes: Sistema compuesto por varios componentes desarrollados de manera independiente pero que al unirlos constituyen dicho sistema.

Iterativo e incremental: Plantea que ese mismo proceso de desarrollo puede tener tantas iteraciones como sean necesarias y que en cada versión se obtendrá un incremento y mejoramiento de las funcionalidades del sistema.

Incluye la especificación de las actividades de cada una de las fases del ciclo de vida de los proyectos del CEIGE teniendo en cuenta los procesos de CMMI² nivel dos para la Universidad de las Ciencias Informáticas (7).

En la presente investigación se hará uso de la fase Desarrollo, la cual está compuesta por varias disciplinas que se estarán utilizando durante el desarrollo de la investigación. El objetivo de esta fase es obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales. En esta fase se ejecutan las disciplinas Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de liberación.

Durante el flujo de trabajo en la disciplina **Modelado del negocio** se emplea el modelo de dominio, no se utiliza modelo de procesos debido a que no existen procesos de negocio definidos. En el modelo de dominio el artefacto que se genera es el modelo conceptual, el cual describe los aspectos del dominio, identificándose los principales elementos físicos o lógicos del negocio que ayuden a entender el problema y que generalmente se presentan como clases. El modelo conceptual explica cuáles son y cómo se relacionan los conceptos relevantes en la descripción del dominio de un problema, identificando atributos y asociaciones existentes entre ellos. Seguidamente en la disciplina **Requisitos**, se identifican los requisitos funcionales y no funcionales con que contará el sistema, se describen los mismos y se elaboran los prototipos de interfaces de usuario asociados a ellos. En la siguiente disciplina

Análisis y diseño, se define el patrón arquitectónico que determina la estructura fundamental del sistema. Se genera el modelo de diseño, artefacto conformado por los diagramas de clases del diseño con estereotipos web y los diagramas de secuencia. Por último se realiza el modelo de datos y se elaboran los diseños de casos de prueba.

Luego en la disciplina **Implementación**, a partir de los resultados del análisis y diseño se elabora el diagrama de componentes, el diagrama de despliegue y se implementan los requisitos funcionales identificados con sus correspondientes interfaces.

² CMMI: por las siglas en inglés de Capability Maturity Model Integration.

En las **Pruebas internas** se realizan pruebas de caja negra mediante la técnica de partición equivalente y las pruebas de caja blanca utilizando la técnica del camino básico, también se resuelven las no conformidades detectadas durante la ejecución de estas pruebas.

1.6 Ambiente de desarrollo

1.6.1 Lenguaje y herramienta para el modelado

UML 2.3: Lenguaje que permitió visualizar, especificar, construir y documentar los artefactos del sistema.

Herramienta CASE³.

Visual Paradigm 8.0:

Visual Paradigm es una herramienta que permitió el modelado del sistema ayudando a una rápida construcción del mismo con calidad. Permitted diseñar los diagramas de clases del diseño, de secuencia, despliegue, de componente, el modelo conceptual y el modelo de datos. Entre sus principales características se encuentra que con fines académicos se puede utilizar sin pago, la compatibilidad entre las diferentes ediciones, además de ser fácil de instalar y actualizar.

Fundamentación de la herramienta de modelado a utilizar

Como herramienta de modelado se selecciona el Visual Paradigm pues permite trabajar de forma colaborativa, hacer un trabajo más organizado y ágil. Es una herramienta multiplataforma fácil de utilizar, posibilitando crear las tablas y sus relaciones en el sistema gestor de base de datos, a partir del modelo de datos permitiendo así ahorrar tiempo durante la implementación. Además actualmente la UCI cuenta con una licencia que permite el uso de esta herramienta.

1.6.2 Lenguajes de programación

Lenguajes del lado del servidor.

Lenguaje PHP:

Se decide utilizar PHP dado a que es un lenguaje de programación usado normalmente para la creación de páginas web dinámicas, es una tecnología de código abierto que resulta muy útil para diseñar de forma rápida. También porque su interpretación y ejecución se realizan en el servidor en el cual se encuentra almacenada la página y el cliente sólo recibe el resultado de la ejecución. PHP tiene la capacidad de ser ejecutado en los sistemas operativos tales como Linux o Windows.

Fundamentación del lenguaje de programación a utilizar

³ Ingeniería de Software Asistida por Ordenador (Computer Aided Software Engineering, CASE por sus siglas en inglés).

Para la implementación del sistema se decide utilizar como lenguaje de programación PHP en su versión 5.4, dado a que el mismo está diseñado para la creación de páginas web dinámicas, es un lenguaje gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. Además se escoge este lenguaje de programación, debido que el marco de trabajo elegido para la implementación del sistema, Symfony2, todo lo compila a código PHP.

Lenguajes del lado del cliente.

HTML:

Es usado en el desarrollo del sistema para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes y la creación de las diferentes vistas del mismo. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>).

1.6.3 Frameworks (Marco de trabajo)

Twitter Bootstrap 3.0:

Se decide utilizar Bootstrap dado a que es un framework de aplicaciones para usuario elegante e intuitivo, y de gran alcance para el desarrollo web más rápido y sencillo. Es rápido, fuerte para el desarrollo web. Además Bootstrap hace uso de determinados elementos HTML y propiedades CSS 3 que requieren el uso del HTML5.

Doctrine 2:

Se utiliza Doctrine por ser un potente y completo sistema ORM⁴ para PHP 5.2+ que incorpora una capa de abstracción a base de datos (DBL⁵). Uno de sus rasgos importantes es la habilidad de escribir opcionalmente las preguntas de la base de datos orientada a objeto.

Funcionalidades que facilitaron la implementación:

- ❖ Exporta una base de datos existente a sus clases correspondientes.
- ❖ Convierte clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos (8).

Symfony2:

Para el desarrollo del sistema en cuestión como marco de trabajo se escoge Symfony2, dado que es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo en sistemas complejos. (20).

Symfony2 está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales, y se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows (20).

⁴ Object Relational Mapper

⁵ Data Base Language

Symfony utiliza el framework ORM llamado Doctrine, el mismo es hogar de varias bibliotecas de PHP, se centra principalmente en el almacenamiento de la base de datos y el mapeo de objetos. Sus componentes básicos son Mapeo entre Objetos Relaciones (ORM) y la capa de abstracción de la base de datos (DBAL). Doctrine es muy estable, con una base de código de alta calidad, un mapeo extremadamente flexible y de gran alcance. Además posee una gran comunidad y se integra con muchos marcos diferentes (13).

Como gestor de plantillas Symfony utiliza Twig su principal característica se encuentra en la herencia de plantillas, también es muy rápido y seguro, permite separar el código PHP del HTML lo que garantiza que los usuarios pueden modificar el diseño de la plantilla, sin modificar el código. Twig es también un framework muy flexible que permite al desarrollador definir sus propias etiquetas y filtros personalizados, y crear su propio DSL (21).

Symfony2 cuenta con ciertas ventajas sobre los demás frameworks en la implementación de un sistema.

- Fácil de instalar y configurar en la mayoría de las plataformas.
- Sigue la mayoría de buenas prácticas y patrones de diseño para la web.
- Código fácil de leer y documentar, además permite un mantenimiento muy sencillo.
- Permite su integración con librerías desarrolladas por terceros.

Fundamentación del marco de trabajo a utilizar

Se considera que el framework más apropiado para construir el sistema es Symfony2 debido a las características antes argumentadas. Este marco de trabajo propone el estilo Modelo - Vista – Controlador, optimiza el desarrollo de las aplicaciones web. Ha sido probado en numerosos proyectos reales y utiliza conceptos exitosos de terceros entre los cuales se encuentra plenamente un framework ORM de los más importantes que existen en PHP llamado Doctrine que se encarga de la comunicación con la base datos, independientemente del Gestor de Base Datos utilizado. Incluye otro framework bastante conocido llamado Twig que constituye un poderoso motor de plantillas que permite separar el código PHP del código HTML, además entre otras cosas una mayor organización en el código y sobre todo realizar el diseño sin tocar el código. Symfony2 contiene un amplio soporte para la seguridad de la información sobre todo contra ataques comunes como SQLInjection y XSS .CSRF. Además tener más conocimiento del mismo que de los otros framework hace que se aproveche más el tiempo para la realización del sistema.

1.6.4 Herramientas de desarrollo

Entorno integrado de desarrollo

Geany:

Como herramienta para la implementación del sistema se utilizó Geany, editor de texto utilizando el toolkit GTK2 con las características básicas de un entorno de desarrollo integrado.

Fue desarrollado para proporcionar un pequeño y rápido IDE, que sólo tiene unas pocas dependencias de otros paquetes. Es compatible con muchos tipos de archivos y tiene algunas características agradables como son:

- El resultado de sintaxis.
- Plegado de código.
- Nombre de símbolo de autocompletar.
- Construir terminación/ snippets.
- Auto cierre de etiquetas XML y HTML.
- Muchos tipos de archivos soportados, incluyendo C, Java, PHP, HTML, Python, Perl, Pascal (lista completa).
- Navegación código.
- Administración simple proyecto.
- Interfaz de complementos.

Geany es sabido ejecutar bajo Linux, FreeBSD, NetBSD, OpenBSD, MacOS X, v5.3 AIX, Solaris Express y Windows. De manera general, se debe ejecutar en todas las plataformas, que es apoyado por las librerías GTK. Sólo el puerto de Windows de Geany le faltan algunas características. El código está disponible bajo los términos de la licencia Pública General GNU (36).

Fundamentación del entorno integrado de desarrollo a utilizar

Se decide utilizar para la implementación el editor de texto Geany, el cual fue diseñado para proporcionar un pequeño y rápido IDE. También debido a todas las características antes mencionadas que ofrece para el uso de PHP como lenguaje de programación elegido y principalmente por las prestaciones de la máquina que se utilizó para la implementación, consume poco recurso, es ágil, pequeño y rápido.

Servidor de Aplicaciones web

Servidor web Apache 2.0:

Se decide utilizar este servidor ya que es una tecnología gratuita de código fuente abierta, puede ser usado en varios sistemas operativos, lo que lo hace prácticamente universal.

Es un servidor altamente configurable, es decir, se pueden elegir qué características van a ser incluidas en el servidor seleccionando qué módulos se van a cargar, ya sea al compilar o al ejecutar el servidor (9).

Sistema Gestor de Base de Datos:

PostgreSQL 9.1:

Se utiliza PostgreSQL, dado que es un servidor de base de datos relacional⁶, libre. Se destaca en ejecutar consultas complejas, consultas sobre vistas, subconsultas y joins de gran tamaño. Permite la definición de tipos de datos personalizados e incluye un modelo de seguridad completo. Como toda herramienta de software libre PostgreSQL tiene entre otras ventajas la de contar con una gran comunidad de desarrollo en Internet, su código fuente está disponible sin costo alguno y es un sistema multiplataforma (1).

1.7 Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (15).

Los patrones de diseño se dividen en dos vertientes, los patrones **GRASP**⁷ y los **GoF**⁸.

Patrones GRASP: Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones (16). Algunos de los patrones GRASP más utilizados son:

- **Experto:** Asigna una responsabilidad al experto en información: la clase que posee la información necesaria para cumplir con la responsabilidad (17).
- **Creador:** Asigna a la clase B la responsabilidad de crear una instancia de clase A (17).
- **Controlador:** Asigna la responsabilidad de administrar un mensaje de eventos del sistema a una clase (17).
- **Bajo acoplamiento:** Asigna responsabilidades de modo que se mantenga bajo acoplamiento (17).
- **Alta cohesión:** Asigna responsabilidad de modo que se mantenga alta cohesión (17).

Patrones GoF: Se dieron a conocer a principios de los años 90 con el libro “Design Patterns. Elements of Reusable Object-Oriented Software”⁹. En dicho libro se hace una recopilación de 23 patrones de diseño comunes, clasificados en tres grupos de acuerdo a su naturaleza:

⁶ **Base de datos relacional:** Modelo utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente basados en el empleo de relaciones entre las tablas (14).

⁷ **GRASP:** por las siglas en inglés de General Responsibility Assignment Software Patterns, en español asignar responsabilidades (20).

⁸ **GoF:** por las siglas en inglés de Gang of Four, en español Banda de los Cuatro.

- **Patrones Creacionales:** Inicialización y configuración de objetos (18).
- **Patrones Estructurales:** Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes (18).
- **Patrones de Comportamiento:** Más que describir objetos o clases, describen la comunicación entre ellos (18).

Algunos de los patrones de diseño GoF son los que se muestran a continuación:

- **Fachada (grupo estructural):** Proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso (17).
- **Factoría (grupo creacional):** Proporciona una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas (17).
- **Singleton (grupo creacional):** Garantiza que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma (17).
- **Mediador (grupo comportamiento):** Encapsula la forma en que interactúan un grupo de objetos, promoviendo así un acoplamiento débil al evitar las referencias explícitas entre los objetos (17).

1.8 Conclusiones del capítulo

Después de realizado el estudio para este primer capítulo se puede arribar a las siguientes conclusiones:

- Se construyó el marco teórico de la investigación, el cual permitió fundamentar conceptos, métodos y herramientas posibilitando el desarrollo del sistema.
- Se realizó un estudio sobre las herramientas y modelos existentes relacionados con la gestión y análisis de las decisiones arquitectónicas, de los cuales la investigación se fortaleció con elementos de algunos de ellos para diseñar la solución propuesta.
- La utilización de los marcos de trabajo, lenguajes y herramientas estudiados como parte del ambiente de desarrollo, facilitaron entre otros aspectos el desarrollo del sistema, ya que se utilizaron tecnologías libres y la obtención de un producto como resultado final con la calidad requerida que permitirá la gestión y análisis de las decisiones arquitectónicas que se tomen en el CEIGE.

⁹ Traducido al español como: Patrones de Diseño. Elementos de software reusable orientado a objetos.

Capítulo 2: Análisis y Diseño de la Solución Propuesta

2.1 Introducción

En el presente capítulo se realiza el análisis de la encuesta aplicada a los arquitectos y especialistas del CEIGE en el tema de decisiones arquitectónicas. Se describen los requisitos funcionales y no funcionales con que contará el sistema. Se muestran los prototipos de interfaz de usuario asociados a cada uno de los requisitos funcionales, los diferentes artefactos generados durante la disciplina Análisis y diseño. Se realiza la validación del diseño mediante métricas, en aras de verificar los atributos de calidad definidos por las métricas seleccionadas.

2.2 Selección del experto humano.

Para que la encuesta aplicada tuviera éxito, fue imperativo que expertos humanos (EH) adecuados estuvieran disponibles y que contaran con las siguientes características:

- Los expertos existentes deben estar posibilitados para resolver problemas en el dominio del tema.
- Deben ser capaces de describir el conocimiento del dominio y cómo se debe aplicar.
- Los expertos deben tener la disposición a dar conocimiento y colaborar en los esfuerzos de desarrollo.
- Los expertos deben disfrutar de buena reputación entre los potenciales usuarios del sistema (38).

Considerando las características que el experto humano debe reunir, se seleccionaron ocho especialistas del CEIGE, los mismos cuentan con más de un año de experiencia y cumplen con las características descritas anteriormente, ellos son:

1-William González Obregón	5-Osmar Leyet Fernández
2-René Bauta Camejo	6-David Martínez Alarcón
3-Omar Antonio Díaz Peña	7-Juan A. Caballero Protelles
4-Ariadna Rendón Artola	8-Yasel A. Romero Piñeiro

Estos especialistas son expertos humanos que poseen las siguientes particularidades:

- Buena capacidad de memoria o de razonamiento.
- Poseen buena cantidad de conocimiento adquirido de años de estudio y práctica, cuya organización y naturaleza es de calidad.
- Han desarrollado la habilidad de percibir grandes patrones de información significativos, de manera que la solución de problemas la llevan a cabo de una forma intuitiva.

2.3 Análisis de la encuesta aplicada

Para el proceso de investigación sobre el tema en cuestión, se realizó una encuesta a un grupo de arquitectos y especialistas capacitados en el tema de Arquitectura de Software, es decir, específicamente en las decisiones arquitectónicas que se toman en sus respectivos proyectos, con un promedio de 3 años de experiencia, de la cual se obtuvieron los siguientes resultados:

Resultados de la encuesta

- Cuando se presenta algún problema arquitectónico en los proyectos se da solución al momento sin consultar otros medios, en ocasiones utilizan su experiencia individual o consultan con otros arquitectos de más experiencia.
- Cuando se le da solución a un problema arquitectónico en algunos proyectos queda la evidencia de la solución tomada, para ello usan el documento de decisiones arquitectónicas en el que se registran las posibles alternativas a considerar y se destacan las más convenientes, algunos lo hacen como parte de la documentación del proyecto. En otros casos mencionan que no se registran las decisiones que se toman, o sea, no existen los artefactos o aplicaciones para dejar las evidencias correspondientes, provocando que la ocurrencia de problemas similares a los ya identificados, lleve consigo un nuevo análisis de una posible solución. Todo ello demuestra que no existe homogeneidad con respecto a las decisiones que se toman, por tal motivo se hace necesario el desarrollo del sistema en cuestión.
- Los que aplican mecanismos para reutilizar decisiones arquitectónicas que se toman en otros proyectos mencionan que:
 1. Consultan con los propios arquitectos del sistema u otros especialistas en el tema.
 2. Consultan documentación donde se registran las decisiones arquitectónicas referentes al proyecto.
 3. Realizan talleres con los involucrados de los proyectos, donde se han tomado decisiones similares, para identificar si es posible reutilizar su decisión y medir el impacto de su aplicación.

Los resultados de la encuesta aplicada dieron a conocer la necesidad de un sistema que ayude a la toma de decisiones en cuanto a las decisiones arquitectónicas, debido a que en los proyectos de gestión de negocios el 70% de los problemas son comunes, esto garantizaría que el 70 % de las soluciones también sean comunes. También permitiría ahorro de tiempo y esfuerzo, además daría garantía de usar una solución previamente documentada, agilizaría el proceso de búsqueda, aportando un valor agregado asociado a la estandarización del registro de las decisiones arquitectónicas. El desarrollo de este sistema brindará un espectro más amplio a los arquitectos de menor experiencia. De forma general la realización de este trabajo garantizaría la homogeneidad de las soluciones. Para consultar la encuesta aplicada ver [Anexo 1](#).

2.4 Técnicas para la Captura de requisitos

Existen varias técnicas para llevar a cabo el proceso de **captura de requisitos**. Es tarea del equipo de desarrollo determinar cuál o cuáles son las más factibles a utilizar para el producto a desarrollar. Entre algunas de estas técnicas se pueden mencionar:

- **Entrevistas:** Es un medio tradicional de obtención de requisitos. La entrevista es un método muy efectivo que permite conocer los problemas de los clientes y encontrar requisitos generales. Para aplicar este método es necesario conocer la forma en que se debe realizar una entrevista para lograr una buena comunicación entre el entrevistador y el entrevistado (23).
- **Prototipos:** Es la representación o visualización de parte del sistema. Es una herramienta valiosa para clarificar requisitos confusos. Proveen a los usuarios un contexto para entender mejor qué información necesitan proporcionar (23).
- **Tormenta de ideas:** Esta técnica es usada para generar nuevas ideas y encontrar la solución a cuestiones específicas. Es muy común en los comienzos del proceso de ingeniería de requisitos (23).

Después de analizar las definiciones de cada una de las técnicas de captura de requisitos anteriormente citadas se llevó a cabo la utilización de cada una de ellas, ejemplo mediante la entrevista, se pudo conocer las necesidades de los clientes que en este caso fueron los arquitectos del centro. Luego se realizaron los prototipos, en donde se les fue mostrando a los clientes para ver según las necesidades que ellos tenían si se cumplía con lo requerido. También mediante tormenta de ideas en conjunto con los profesionales implicados en el tema en cuestión, contribuyó a conformar nuevas ideas para la realización del sistema.

2.5 Requisitos de software

Los requisitos funcionales muestran las características requeridas por el sistema informático propuesto, que expresan una capacidad de acción del mismo, una funcionalidad o comportamiento interno; generalmente expresada en una declaración en forma verbal (37). A continuación se describen los requisitos funcionales y no funcionales del sistema.

2.5.1 Requisitos Funcionales

Durante la disciplina de Requisitos se identificaron 59 Requisitos funcionales para el sistema de gestión y análisis de las decisiones arquitectónicas en el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP.

RF1: Gestionar decisión

RF1.1 Adicionar decisión

RF1.2 Modificar decisión

RF1.3 Eliminar decisión

RF1.4 Listar decisiones

RF1.5 Mostrar decisión

RF1.6 Exportar decisión

RF1.7 Enviar notificación con decisiones

RF2: Gestionar departamento

RF2.1 Adicionar departamento

RF2.2 Modificar departamento

RF2.3 Eliminar departamento

RF2.4 Listar departamentos

RF2.5 Mostrar departamento

RF3: Gestionar proyecto

RF3.1 Adicionar proyecto

RF3.2 Modificar proyecto

RF3.3 Eliminar proyecto

RF3.4 Listar proyectos

RF3.5 Mostrar proyecto

RF4: Gestionar situación

RF4.1 Adicionar situación

RF4.2 Modificar situación

RF4.3 Eliminar situación

RF4.4 Listar situaciones

RF4.5 Mostrar situación

RF4.6 Buscar situaciones semejantes

RF4.7 Exportar situaciones semejantes

RF4.8 Enviar notificación con situaciones semejantes

RF5: Gestionar disciplina

RF5.1 Adicionar disciplina

RF5.2 Modificar disciplina

RF5.3 Eliminar disciplina

RF5.4 Listar disciplinas

RF5.5 Mostrar disciplina

RF6: Gestionar actividad

RF6.1 Adicionar actividad

RF6.2 Modificar actividad

RF6.3 Eliminar actividad

RF6.4 Listar actividades

RF6.5 Mostrar actividad

RF7: Gestionar artefacto

RF7.1 Adicionar artefacto

RF7.2 Modificar artefacto

RF7.3 Eliminar artefacto

RF7.4 Listar artefactos

RF7.5 Mostrar artefacto

RF8: Gestionar estado

RF8.1 Adicionar estado

RF8.2 Modificar estado

RF8.3 Eliminar estado

RF8.4 Listar estados

RF8.5 Mostrar estado

RF9: Gestionar fase

RF9.1 Adicionar fase

RF9.2 Modificar fase

RF9.3 Eliminar fase

RF9.4 Listar fases

RF9.5 Mostrar fase

RF10: Gestionar usuario

RF10.1 Adicionar usuario

RF10.2 Modificar usuario

RF10.3 Eliminar usuario

RF10.4 Listar usuarios

RF10.5 Mostrar usuarios

RF11: Generar reportes de situaciones semejantes

RF12: Generar reportes de decisiones

RF13: Listar situaciones por proyecto

RF14: Listar decisiones por proyecto

2.5.2 Requisitos no Funcionales

Los requisitos no funcionales son propiedades o cualidades que debe tener el producto. Estas propiedades son las características que hacen al producto atractivo, usable, rápido o confiable, por solo mencionar algunas.

Durante el proceso de captura de requisitos se identificaron algunos requisitos no funcionales (**RnF**) los cuales se listan a continuación:

Usabilidad

RnF1: El sistema presentará una interfaz legible, simple de usar e interactiva.

Seguridad

RnF2:

- El sistema debe garantizar protección contra acciones no autorizadas o que puedan afectar la seguridad de los datos.
- El sistema debe mostrar las funcionalidades de acuerdo a los permisos del usuario que esté activo.
- Debe verificar que el usuario esté autenticado antes de que pueda realizar alguna acción sobre el sistema.

RnF3: Para la utilización del sistema se recomienda

- Navegador Mozilla Firefox 26.0 o superior
- Chromium Browser 30.0 o superior
- Sistema operativo Windows 7 o superior
- Sistema operativo Ubuntu 12.04

Hardware

RnF4:

Requerimientos mínimos para la conexión del cliente:

- PC con un sistema operativo previamente instalado y un navegador web para acceder a la aplicación.

Requerimientos mínimos para la conexión del servidor:

- PC con sistema operativo Ubuntu 12.04 con disco duro mínimo de 40 Gb, 1Gb de RAM, micro a 1.7 GHz.

2.6 Técnicas para la Validación de Requisitos

Existen varias **técnicas para validar los requisitos**, las cuales se aplican con el objetivo de examinar las especificaciones para asegurar que todos los requisitos han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones y que los errores detectados hayan sido corregidos (22). Las siguientes técnicas mencionadas fueron usadas en la investigación:

- **Revisiones:** Los requisitos son analizados sistemáticamente por un equipo de revisores, en busca de anomalías y/u omisiones (24). En este caso el equipo de revisores fueron los clientes del sistema, que fueron dando su criterio sobre los problemas encontrados.
- **Prototipos:** Se muestra un modelo ejecutable del sistema a los usuarios finales y los clientes, para que puedan ver si dicho modelo cumple con sus necesidades reales (24). Una vez conformado el modelo, los mismos fueron presentados a los clientes y estos arrojaron que cumplían con sus necesidades.

2.7 Modelo de Diseño

Diagramas de clases del diseño con estereotipos web

Los diagramas de clases especifican las diferentes clases que serán utilizadas en el sistema y las relaciones que existen entre ellas. A continuación se muestra el diagrama de clases perteneciente al requisito Gestionar decisión y una breve descripción del mismo. En el [Anexo 2](#), [Anexo 3](#) y [Anexo 4](#) se pueden consultar otros diagramas elaborados.

RF1: Gestionar decisión

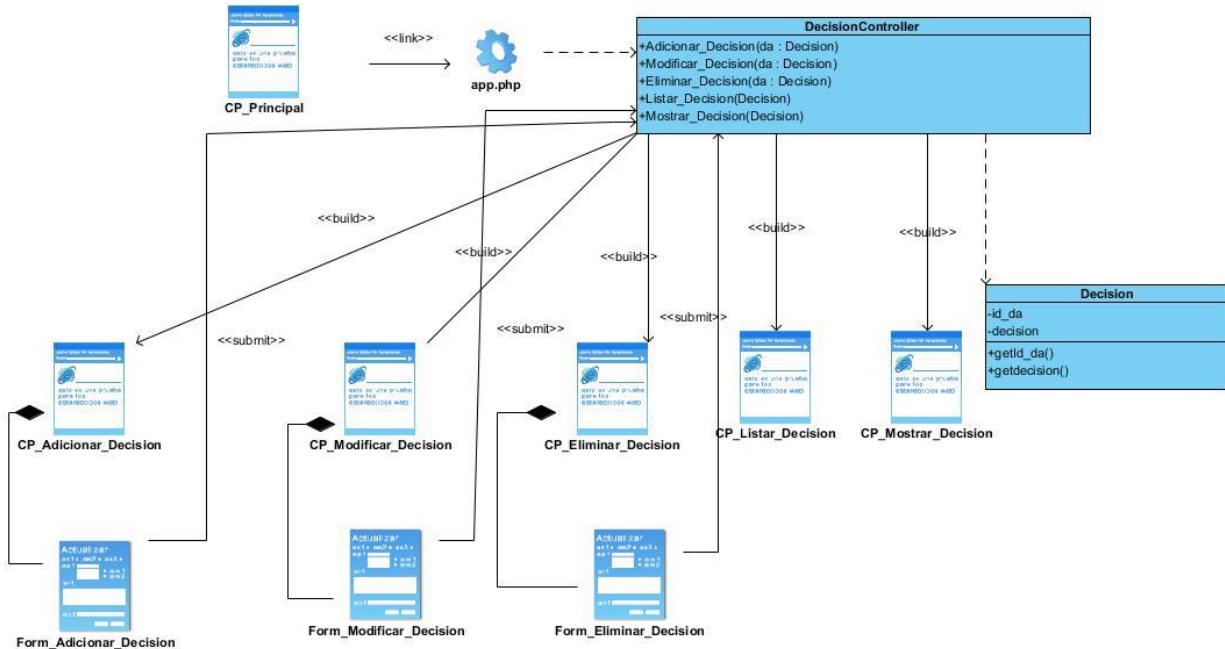


Figura 1: Gestionar decisión.

Las **Client page** representan páginas web encargadas de mostrar los formularios de información al usuario. El **Server page**, servidor que se encarga de direccionar según las peticiones que haga el usuario.

La clase **DecisiónController** es la encargada de manejar la comunicación entre la vista y la clase modelo Decisión.

En el modelo se puede observar la **Client page** principal, primeramente se selecciona la opción Decisión y según la funcionalidad que se ejecute redirecciona hacia la **Client page** indicada, luego se construye el formulario correspondiente, el cual al introducir los datos válidos los envía a la clase **DecisiónController**, encargándose esta del proceso de enviar las acciones a la base de datos.

Diagrama de secuencia

Los diagramas de secuencia describen la interacción entre los objetos de una aplicación y los mensajes recibidos y enviados por los objetos (25).

En el presente epígrafe se muestra el diagrama de secuencia perteneciente al requisito Adicionar decisión y una breve descripción del mismo. En el [Anexo 5](#), [Anexo 6](#) y [Anexo 7](#) se pueden consultar otros diagramas elaborados.

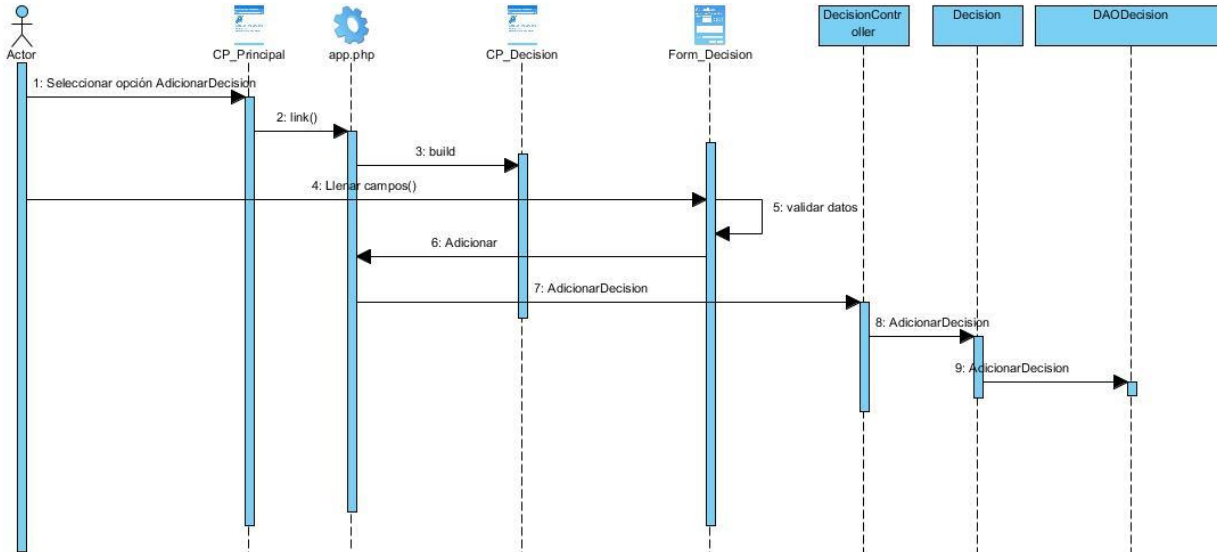


Figura 2: Adicionar decisión.

El siguiente diagrama muestra un determinado escenario, los eventos generados por actores externos, su orden y los eventos internos del sistema. Contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

2.8 Modelo de datos

Otra de las etapas del diseño es la elaboración del **modelo de datos**, con el propósito de garantizar que los datos persistentes sean almacenados coherentemente y definir el comportamiento que debe ser implementado en la base de datos (26).

A continuación se muestran las tablas correspondientes al modelo de datos y una breve descripción del mismo.

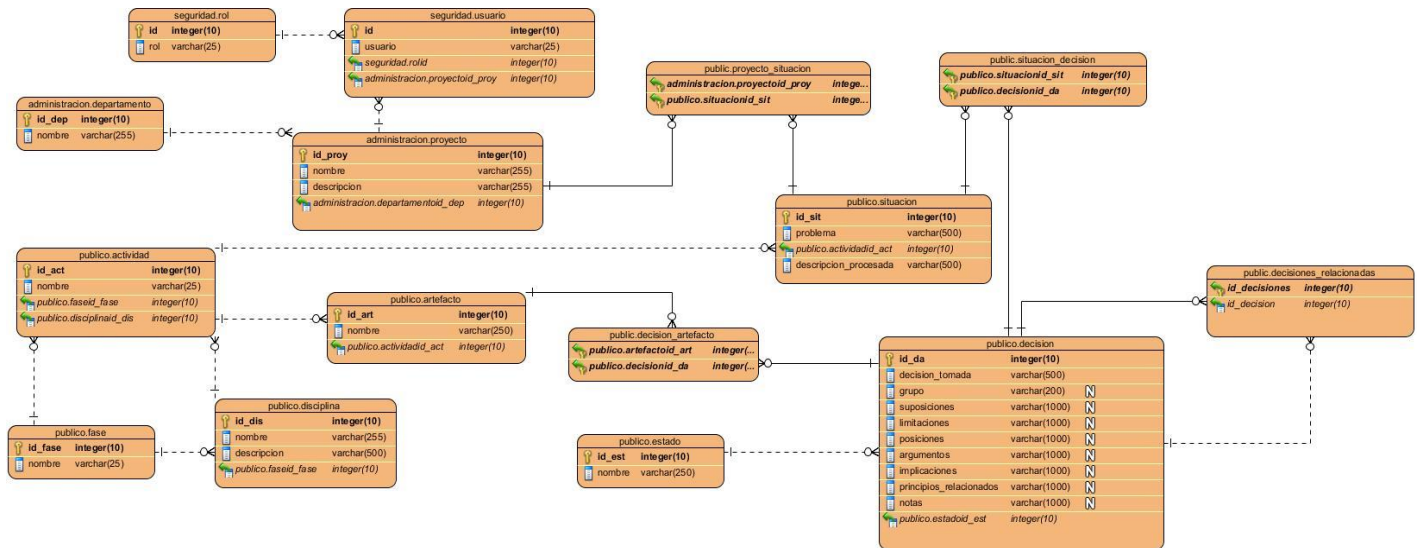


Figura 3: Modelo de datos.

El modelo de datos propuesto en la solución cuenta con un total de 15 tablas, y entre las más significativas se encuentran decisión y situación. De las restantes 13 tablas administración_proyecto, administración_departamento, seguridad_usuario y seguridad_rol, se encargan de gestionar los datos de los proyectos, departamentos, usuarios y roles en el sistema.

2.9 Patrones de diseño utilizados

A continuación se enuncian los diferentes patrones de diseño utilizados durante el proceso de desarrollo de la solución. Los mismos son propuestos por Symfony 2, que fue el marco de trabajo sobre el cuál se trabajó. A continuación se ejemplificará cuáles son y cómo son utilizados.

Patrones generales de software para asignación de responsabilidades (GRASP)

Creador: En las clases controladoras se encuentran todas las acciones definidas para el módulo **AdministracionBundle**. En las acciones se crean los objetos de las clases que representan las entidades, evidenciando de este modo que las clases controladoras son creadoras de dichas entidades.

Experto: Es uno de los más utilizados, puesto que Symfony 2 utiliza el ORM Doctrine 2 para realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

Alta cohesión: Symfony 2 permite asignar responsabilidades con una alta cohesión ya que los controladores definen las acciones para las plantillas y colaboran con otras clases para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.

Controlador: Todas las peticiones web son manejadas por un solo controlador frontal **app.php**, que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL solicitada por el cliente.

Bajo acoplamiento: Las clases controladoras heredan solamente de **BaseController.php** para lograr un bajo acoplamiento de clases.

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

2.10 Validación del diseño mediante métricas

Antes de comenzar la fase de implementación es necesario comprobar la calidad del diseño elaborado, para ello se emplean métricas. Las mismas son una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Se emplean las métricas Tamaño operacional de la clase (TOC) y Relaciones entre clases (RC), diseñadas para evaluar los siguientes atributos de calidad:

- **Responsabilidad:** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.
- **Complejidad de implementación:** Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

- **Reutilización:** Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** Consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy ligada a la característica de Reutilización.
- **Complejidad del mantenimiento:** Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas:** Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

2.10.1 Métrica Tamaño Operacional de Clase

Tamaño operacional de clase (TOC): Está dado por el número de métodos asignados a una clase.

A continuación se muestra una serie de tablas enfocadas a un mejor entendimiento de la utilización de esta métrica.

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de Implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Tabla 1: Métrica Tamaño Operacional de Clase.

Atributo	Categoría	Criterio
Responsabilidad	Baja	\leq Promedio
	Media	Entre Promedio y 2^* Promedio
	Alta	$> 2^*$ Promedio
Complejidad de Implementación	Baja	\leq Promedio
	Media	Entre Promedio y 2^* Promedio
	Alta	$> 2^*$ Promedio
Reutilización	Baja	$> 2^*$ Promedio
	Media	Entre Promedio y 2^* Promedio
	Alta	\leq Promedio

Tabla 2: Rango de valores para la métrica TOC.

2.10.2 Métrica Relaciones entre Clases

Relaciones entre clases (RC): Está dada por el número de relaciones de uso de una clase con otras. A continuación se muestran las tablas 3 y 4 encaminadas a un mejor entendimiento de la utilización de esta métrica.

Atributo que afecta	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del acoplamiento de la clase.
Complejidad del mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 3: Atributos de calidad evaluados por la métrica RC.

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

Tabla 4: Rango de valores para la métrica RC.

2.10.3 Resultados de la aplicación de la métrica TOC

Ver los resultados para la métrica TOC en el [Anexo 8](#). Para determinar el valor de los atributos, se calcula el promedio de procedimientos por clases, en este caso se obtuvo como resultado 6,98 aproximándolo al valor entero 7.

En el siguiente gráfico se muestra la representación de los resultados obtenidos agrupados en los intervalos definidos. El mismo refleja que la mayoría de las clases tienen de 1 a 5 procedimientos. Esto demuestra que el funcionamiento general del sistema está distribuido equitativamente entre las diferentes clases.

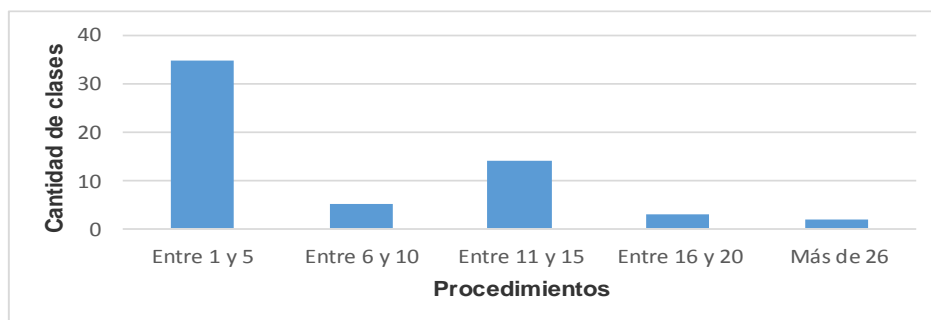


Figura 4: Representación de la evaluación de la métrica TOC.

En la siguiente figura se muestran los resultados obtenidos en el instrumento en por ciento agrupados en los intervalos definidos.

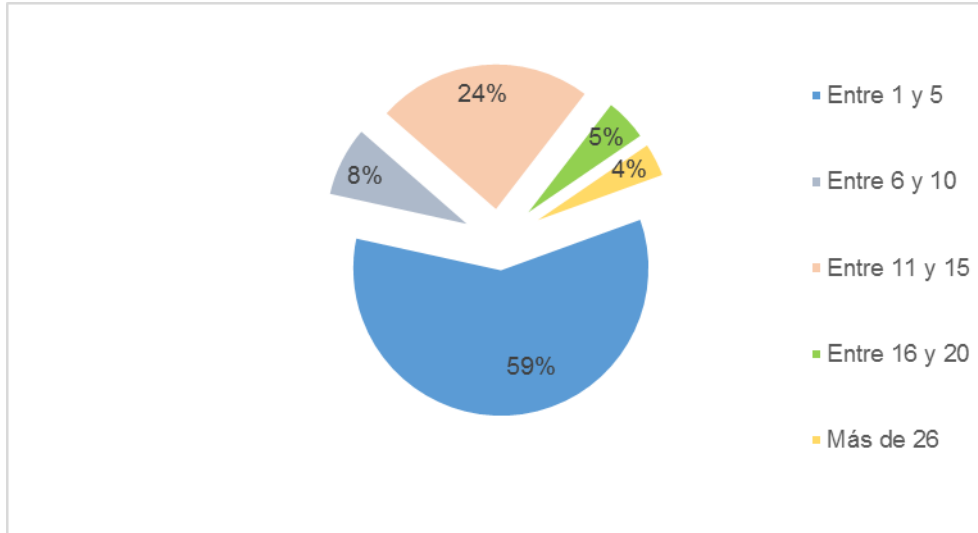


Figura 5: Representación en porcentaje de los resultados obtenidos en la evaluación de la métrica TOC.

En la siguiente figura se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad. Este gráfico muestra un resultado satisfactorio pues el 60% de las clases poseen una baja responsabilidad. Esta característica permite que en caso de fallos, como la responsabilidad está distribuida de forma equilibrada ninguna clase es demasiado crítica como para dejar al sistema fuera de servicio.

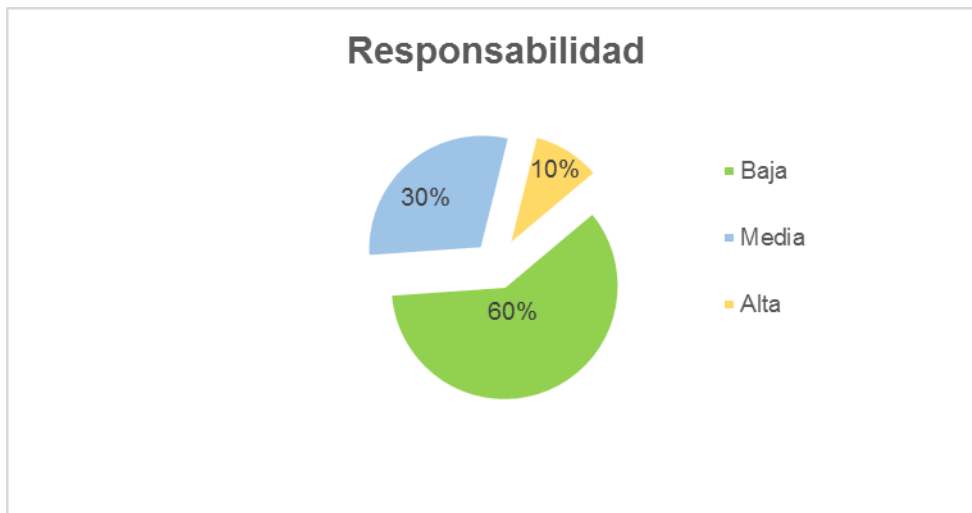


Figura 6: Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.

En la siguiente figura se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo complejidad de implementación. Este gráfico muestra un resultado satisfactorio pues el 60% de las clases poseen una baja complejidad de implementación. Esta característica permite mejorar el mantenimiento y soporte de estas clases.

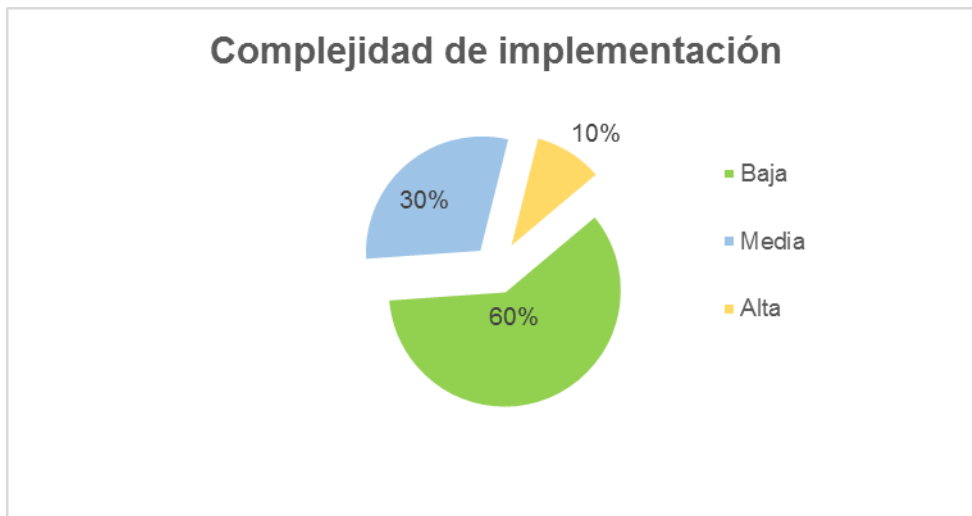


Figura 7: Resultados de la evaluación de la métrica TOC para el atributo complejidad de implementación.

En la siguiente figura se muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo reutilización. El diseño del sistema tiene un grado de eficiencia aceptable pues solamente el 10% del total de las clases poseen una baja reutilización.

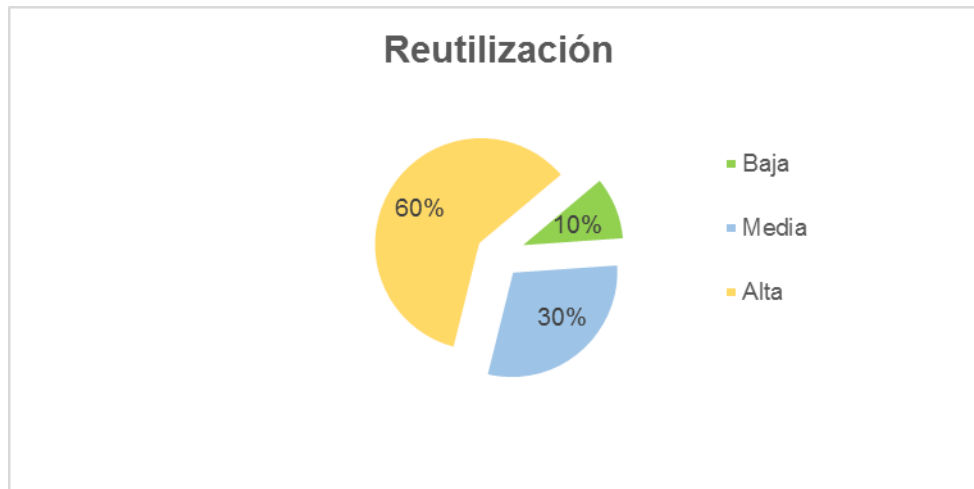


Figura 8: Resultados de la evaluación de la métrica TOC para el atributo reutilización.

Analizando los resultados obtenidos de la métrica TOC, se puede concluir que el diseño del sistema tiene una calidad aceptable teniendo en cuenta los resultados arrojados por los atributos analizados. Solo el 10% de las clases presentan una alta responsabilidad, una alta complejidad de implementación y una baja reutilización, lo cual demuestra que el resultado es satisfactorio.

2.10.4 Resultados de la aplicación de la métrica RC

Ver los resultados para la métrica RC en el [Anexo 9](#). El promedio utilizado para evaluar el criterio es el resultado del cálculo del promedio de la columna Cantidad de relaciones entre clases, en este caso el promedio es 1,05 que se aproximó a 1.

El gráfico de la siguiente figura refleja que el 47% de las clases tienen cero dependencias, y el 19% una dependencia con otra clase. Este resultado es positivo pues demuestra que el 66% de las clases se encuentran dentro de los niveles aceptables de calidad.

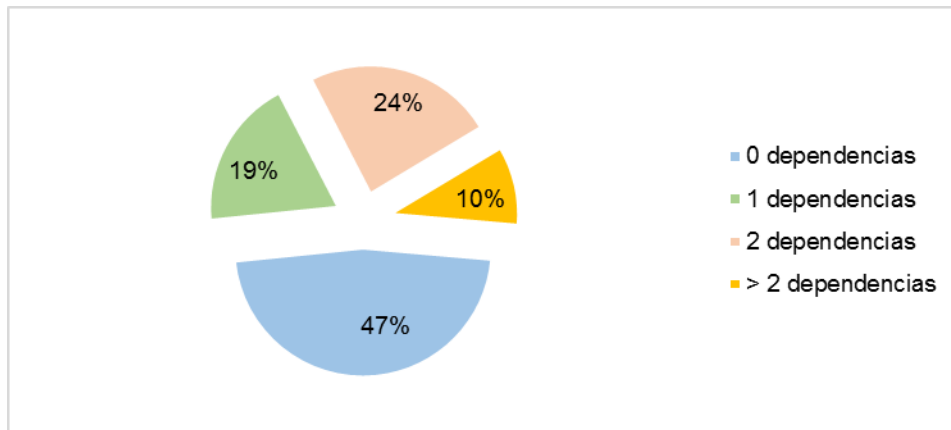


Figura 9: Representación en por ciento de los resultados obtenidos en los intervalos definidos según la métrica RC.

En la siguiente figura se muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento. Se evidencia un bajo acoplamiento entre las clases pues el 47% de las clases no presentan dependencias con otra y el 19% una dependencia con otra. Este resultado es muy favorable para el diseño del sistema pues al existir poca dependencia entre las clases aumenta el grado de reutilización del sistema.

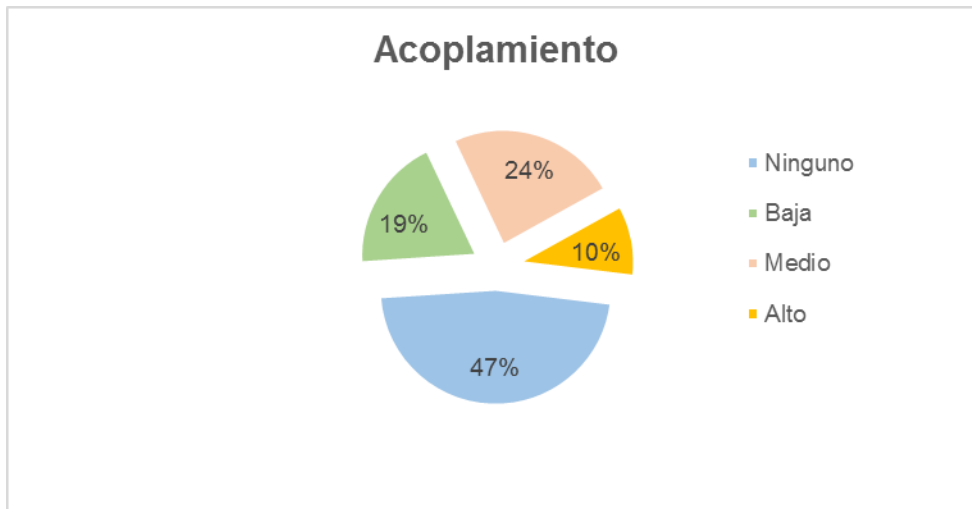


Figura 10: Resultados de la evaluación de la métrica RC para el atributo acoplamiento.

En la siguiente figura se muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. El gráfico refleja un resultado aceptable del atributo pues el 66% de las clases presentan una baja complejidad de mantenimiento.

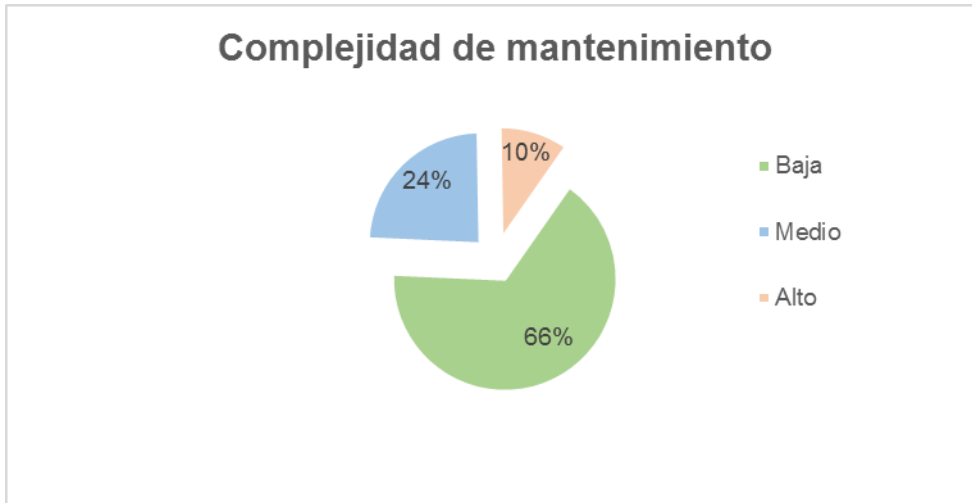


Figura 11: Resultados de la evaluación de la métrica RC para el atributo complejidad de mantenimiento.

En la siguiente figura se muestra la representación de la incidencia de los resultados de la evaluación del atributo reutilización. Esto evidencia que el 66% de las clases poseen una alta reutilización lo que es un factor fundamental que debe ser tenido en cuenta en el desarrollo de software.

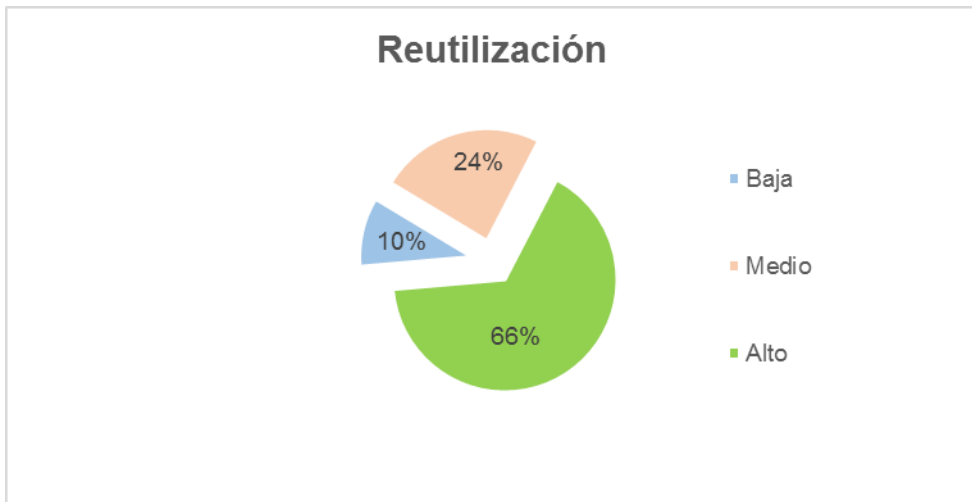


Figura 12: Resultados de la evaluación de la métrica RC para el atributo reutilización.

En la siguiente figura se muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas.

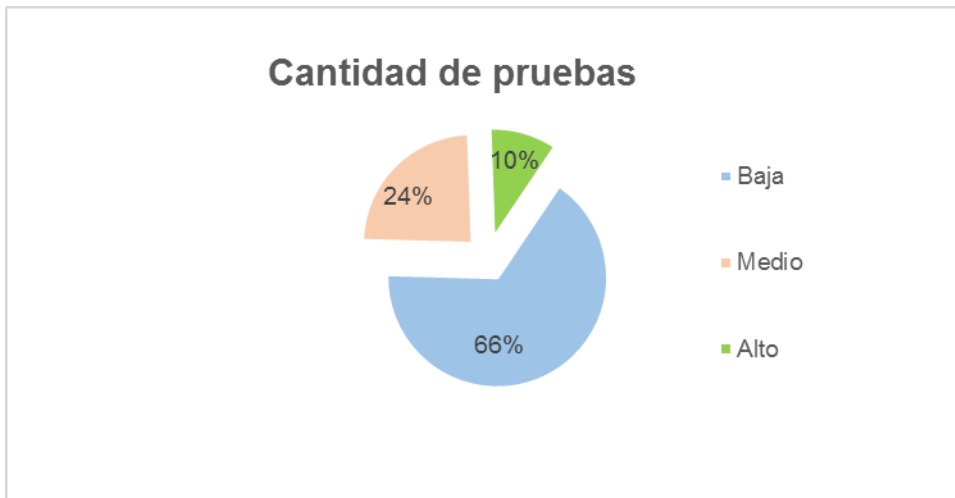


Figura 13: Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del sistema de gestión de análisis de las decisiones arquitectónicas tiene una calidad aceptable.

El 90% de las clases que conforman el sistema poseen menos de tres dependencias con otras clases. Los atributos de calidad se encuentran en un nivel satisfactorio; en el 66% de las clases el nivel de acoplamiento es mínimo.

La complejidad de mantenimiento y la cantidad de pruebas son bajas a un 66%, lo que representa valores favorables para el diseño realizado. Así mismo, existe un alto grado de reutilización al 66%, comportamiento también favorable para este atributo de calidad.

2.11 Conclusiones del capítulo

En el presente capítulo se enunciaron los aspectos fundamentales que se llevan a cabo durante el proceso de análisis y diseño del sistema para la gestión y análisis de las decisiones arquitectónicas que se tomen en el Sistema Integral de Gestión Xedro-ERP, en el mismo se definen las siguientes conclusiones:

- Se realizó una encuesta a un grupo de especialistas capacitados en el tema de Arquitectura de Software, específicamente en las decisiones arquitectónicas que se toman en sus respectivos proyectos, el cual permitió tener un mayor conocimiento de los problemas existentes hoy en el centro, lo que arrojó como conclusión la necesidad del sistema propuesto.
- Se describieron los requisitos funcionales y no funcionales, propiciando la identificación de las posibles funcionalidades con las que contará el sistema.

- Se elaboraron los diagramas de clases y de secuencia correspondientes al sistema. Se construyó el modelo de datos con el propósito de hacer persistir los datos de manera coherente y eficaz y dar paso a la implementación de la base de datos.
- Se validó el diseño propuesto mediante las métricas Tamaño Operacional de Clases y Relaciones entre Clases, que permitieron demostrar que el sistema cuenta con un diseño satisfactorio.

Una vez concluido el análisis y diseño de la solución propuesta puede darse paso al flujo de implementación y pruebas de la misma.

Capítulo 3: Implementación y Prueba

3.1 Introducción

En el presente capítulo se describe la arquitectura del sistema, se muestra el diagrama de componentes que representa los distintos componentes de la solución implementada, así como el de despliegue que permite modelar las relaciones físicas de los distintos elementos que componen el sistema. Se describen las pruebas realizadas al sistema, que tienen como objetivo garantizar la calidad del mismo y el total cumplimiento de los requisitos establecidos con el cliente.

3.2 Descripción de la arquitectura

Para la implementación de la solución se propone utilizar el estilo en capas (capa de presentación, capa de negocio, capa de acceso a datos y capa de datos) y el patrón MVC. El mismo separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones. El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes del tipo de gestor de bases de datos utilizado por la aplicación (18). En la siguiente figura se puede presenciar el uso de este patrón según Symfony 2.

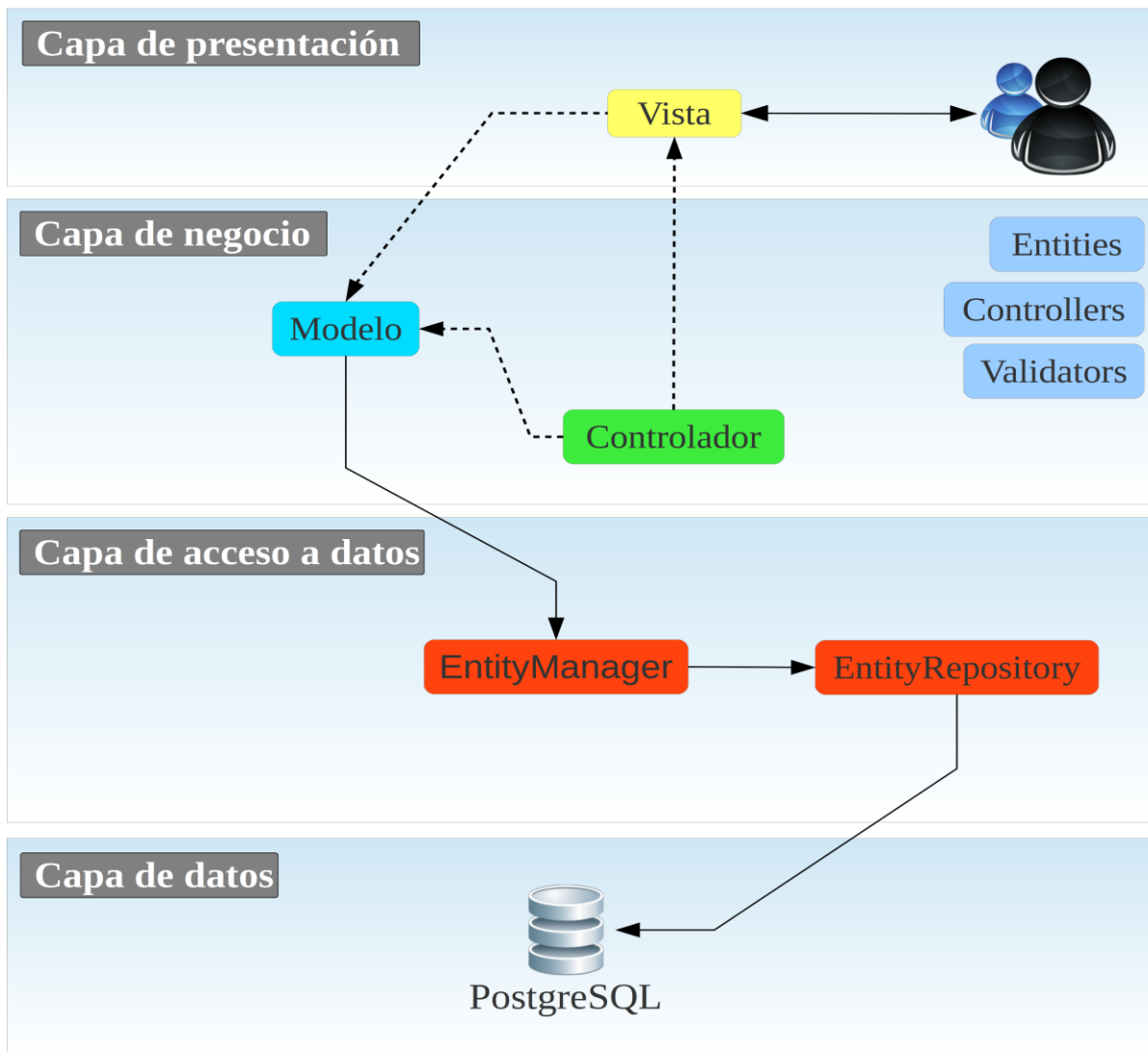


Figura 14: Diseño arquitectónico.

3.3 Diagrama de componentes

Un diagrama de componentes muestra la organización y las dependencias entre un conjunto de componentes. Puede ser un tipo especial de diagrama de clases que se centra en los elementos físicos del sistema y sus relaciones (35). A continuación se muestra el diagrama de componentes elaborado y una breve descripción del mismo.

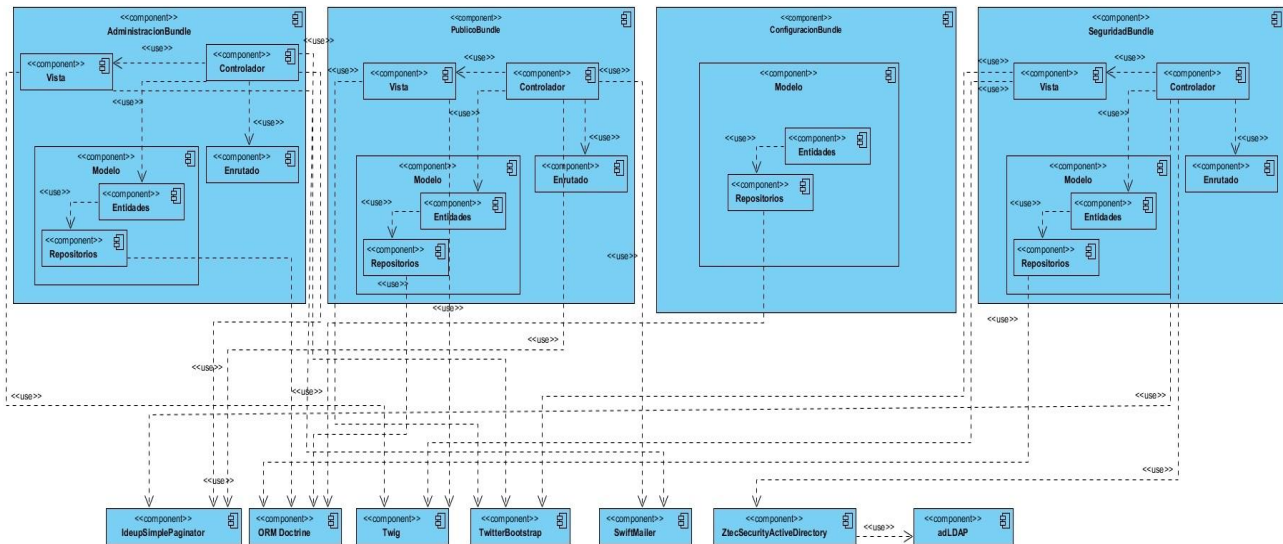


Figura 15: Diagrama de componentes.

Para representar los distintos componentes de la solución implementada se diseñó el siguiente diagrama que justifica la utilización del patrón arquitectónico Modelo – Vista – Controlador. El mismo está compuesto por 4 componentes principales y a su vez cada uno de ellos está asociado a otros componentes, ejemplo de ello “**AdministraciónBundle**”, el cual está estructurado para contener la parte de administración, o sea, maneja todo lo relacionado con el sistema, contiene las entidades (Departamento, Proyecto, Usuario), hace uso de los componentes (IdupSimplePaginador, ORM Doctrine, Twig, TwitterBootstrap, SwiftMailer). “**PúblicoBundle**” es el que contiene a todas las restantes entidades del sistema y se relaciona con los componentes mencionados anteriormente. “**ConfiguraciónBundle**”, hace uso de ORM Doctrine y por último “**SeguridadBundle**”, es el que garantiza la seguridad del sistema, haciendo uso de los componentes (IdupSimplePaginador, ORM Doctrine, Twig, TwitterBootstrap, SwiftMailer, StecSecurityActiveDirectory y adLDAP).

3.4 Aspectos importantes de la implementación del sistema

Durante el proceso de implementación entre las funciones más relevantes se encuentra Buscar situaciones semejantes, dicho método permite establecer un porcentaje de similitud entre las situaciones arquitectónicas a partir de la técnica de minería de texto Análisis Semántico Latente (LSA por sus siglas en inglés). Este algoritmo emplea funciones matemáticas como la Descomposición de Valores Singulares y el coseno del ángulo entre vectores para encontrar el porcentaje de similitud antes mencionado, a continuación se fundamenta dicho proceso. El sistema genera una matriz de situaciones por términos,

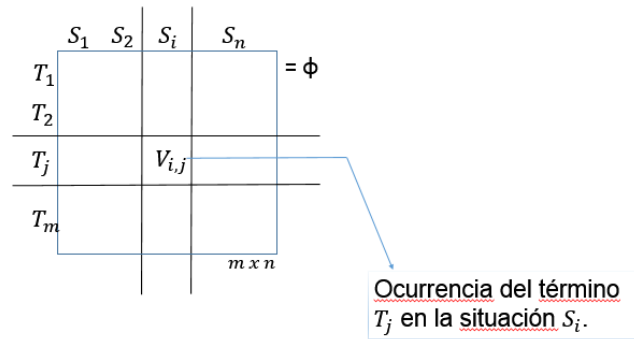


Figura 16. Matriz de situaciones por términos.

al cual se le aplica la técnica de minería de texto Análisis Semántico Latente, el mismo utiliza el algoritmo Descomposición de Valores Singulares arrojando 3 matrices como resultado,

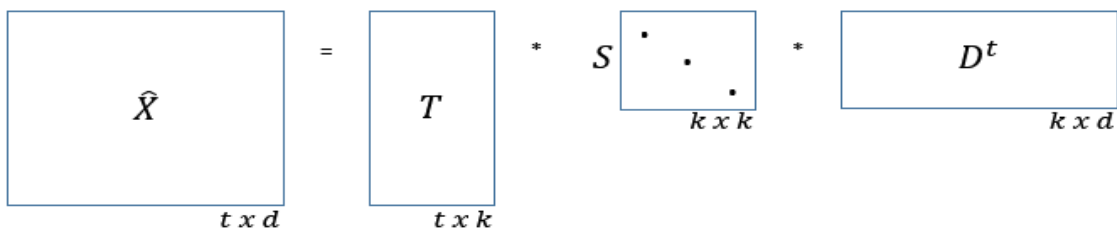


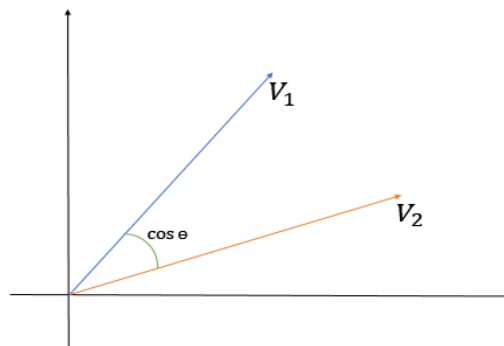
Figura 17. Representación en matrices del algoritmo Descomposición de Valores Singulares.

el paso siguiente es obtener el resultado de la multiplicación de dichas matrices resultantes del proceso anterior como muestra la figura, luego ese resultado se multiplica por su transpuesta obteniendo la matriz final requerida.

$$\hat{X}^t \hat{X}$$

Luego con esa matriz final se seleccionan las columnas (la misma en forma de vectores), para encontrar la

similitud utilizando el $\cos \theta = \frac{v_1 v_2}{\|v_1\| \|v_2\|}$.



Para ver el resultado arrojado de aplicar estas técnicas ver [Anexo 11](#).

3.5 Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Muestra la configuración de los elementos de hardware (nodos) y cómo los elementos y artefactos del software se trazan entre estos. Permite modelar las relaciones físicas de los distintos elementos que componen un sistema y el reparto de los componentes sobre dichos nodos (34).

En el sistema, el usuario accede, desde su puesto de trabajo al sistema que se encuentra instalado en el servidor de aplicaciones web. Luego dicho servidor se conecta a los servidores de base de datos mediante el protocolo TCP¹⁰ para realizar las consultas y obtener los resultados deseados.

A continuación se muestra el diagrama de despliegue.

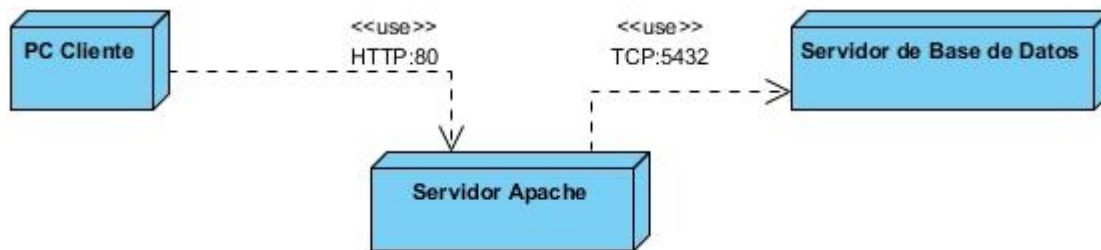


Figura 18: Diagrama de despliegue.

3.6 Pruebas de software

Pressman, declara que las pruebas de software son un elemento crucial para garantizar la calidad del producto y permiten validar las especificaciones, el diseño y la programación. Estas tienen como objetivo, además de descubrir errores, medir el grado en que el software cumple con los requerimientos definidos (22). Existen dos vertientes de pruebas, las pruebas de Caja negra y las pruebas de Caja blanca.

Pruebas de Caja negra: Denominada también pruebas de comportamiento o pruebas funcionales, permiten obtener conjuntos de entradas que ejerciten los requisitos funcionales del software, complementándose con las pruebas de caja blanca, obteniendo errores en las siguientes categorías (22).

- Funciones incorrectas o inexistentes.
- Errores en la interfaz.
- Errores en la estructura de datos.
- Rendimiento.
- Inicialización y terminación.

Para las pruebas de caja negra existen varias técnicas, algunas de ellas son:

¹⁰ Transmission Control Protocol

Partición Equivalente: Permite examinar los valores válidos e inválidos de las entradas existentes en el software. Además descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. Esto permite reducir el número de casos de prueba a elaborar (22).

Análisis de valores límites: Los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta es una técnica que complementa la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, lleva a elección de casos de prueba en los extremos de la clase (22).

Prueba de comparación: Este tipo de pruebas se emplea cuando la fiabilidad del software es algo crítico, (por ejemplo cuando se desarrolla para aeronaves o plantas nucleares) varios equipos de ingeniería del software desarrollan versiones independientes de la misma aplicación, usando los mismos requisitos. Todas las versiones son probadas con los mismos datos, para asegurar que todas proporcionan una salida idéntica (22).

Prueba de la tabla ortogonal: Esta prueba puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para solicitar pruebas exhaustivas. El método de la tabla ortogonal es útil al encontrar errores asociados con fallos localizados (22).

Analizadas estas técnicas se concluye que son útiles siempre que se escoja bien el contexto en el cual se van a aplicar. De estas, la seleccionada fue partición equivalente la cual permite examinar los valores válidos e inválidos de las entradas existentes en el software. Para la aplicación de esta técnica se realizan los diseños de casos de prueba los cuales se basan en una evaluación de las clases de equivalencia para una condición de entrada.

Otro tipo de **pruebas** que se aplica con frecuencia al software son las de **caja blanca**, este es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba (25). Mediante los métodos de prueba de caja blanca:

- Se garantiza que se recorra por lo menos una vez los caminos independientes de cada módulo (22).
- Se ejecuten todas las decisiones lógicas en sus opciones verdadera y falsa (22).
- Se ejerciten todos los bucles en sus límites (22).
- Se usen las estructuras internas de datos para asegurar su validez (22).

Existen varias técnicas de pruebas de caja blanca, algunas de ellas son:

Camino básico: Permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

Para obtener el conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los casos de pruebas obtenidos garantizan que se ejecute al menos una vez cada sentencia del programa (22).

Prueba de condición: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. El propósito de esta técnica es detectar no solo errores en las condiciones, sino también otro tipo de errores (22).

Pruebas de flujos de datos: Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variantes del programa (22).

Prueba de bucles: Se centra exclusivamente en la validez de las construcciones de bucles. Se pueden definir cuatro clases diferentes de bucles: bucles simples, bucles concatenados, bucles anidados y bucles no estructurados (22).

Inicialmente puede parecer que una prueba de caja blanca profunda nos puede llevar a tener programas correctos, definiendo todos los caminos lógicos, generar casos de prueba que examinen exhaustivamente la lógica del programa. Lamentablemente, estas pruebas incluso para programas pequeños el número de caminos lógicos que genera puede ser enorme. Las pruebas de caja blanca no se deben desechar como impracticables, se deben escoger una serie de caminos importantes a ejecutar (22). La técnica optada fue camino básico, dado que permite obtener una medida de la complejidad lógica del diseño y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

3.6.1 Resultados de las pruebas

Pruebas de Caja blanca: La técnica utilizada para la aplicación de esta prueba fue la del camino básico. A continuación se enumeran las sentencias de código del método `semejanzaAction(Request $request)` a modo de ejemplo, el mismo se encarga de mostrar la vista con las situaciones y el porcentaje de semejanza que tienen estas con la situación seleccionada.

```

public function semejanzaAction(Request $request)
{
    $em = $this->getDoctrine()->getManager();//1
    $id = $request->query->get('id');//1
    $comparaciones = array();//1
    $situacion_comparar = $em->getRepository('TesisPublicoBundle:Situacion')->findBy(array('id' => $id));//1
    $situacion = $situacion_comparar[0]->getDescripcionProcesada();//1
    $matrizuno = split(' ', $situacion);//1
    $situaciones = $em->getRepository('TesisPublicoBundle:Situacion')->findAll();//1
    foreach ($situaciones as $lista) {//2
        $temp = $lista->getDescripcionProcesada();//3
        $matrizdos = split(' ', $temp);//3
        $entity = $em->getRepository('TesisPublicoBundle:Situacion')->findBy(array('id' => $lista->getId()));//3
        array_push($comparaciones, array('entidad' => $entity, 'valor' => $this->get_lcs($matrizuno, $matrizdos));//3
    }
    foreach ($comparaciones as $key => $row) {//4
        $entity[$key] = $row['entidad'];//5
        $valor[$key] = $row['valor'];//5
    }
    array_multisort($valor, SORT_DESC, $comparaciones);//6
    return $this->render('TesisPublicoBundle:Situacion:semejante.html.twig', array(
        'situacion' => $situacion_comparar,
        'entities' => $comparaciones,
    ));//6
}

```

Figura 19: Código fuente de la funcionalidad semejanzaAction(Request \$request).

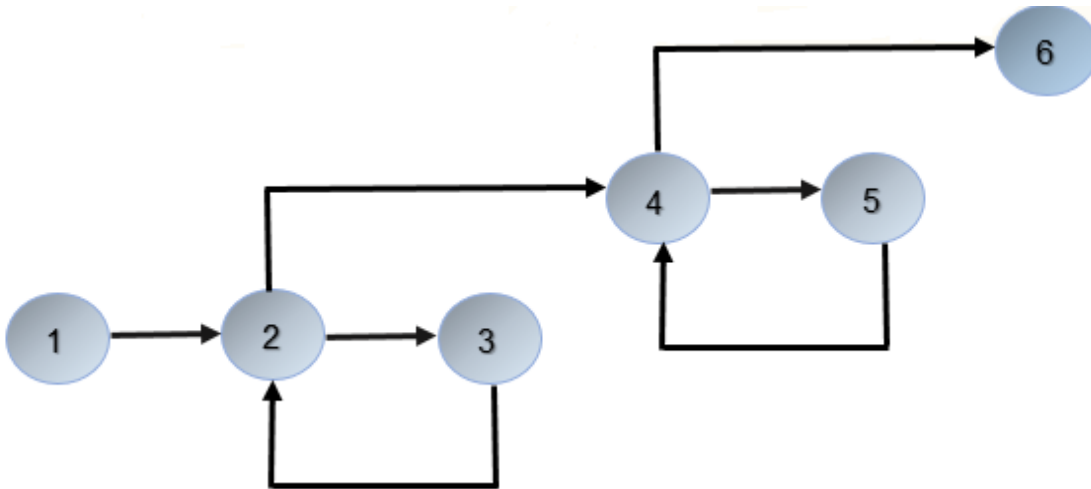


Figura 20: Grafo de flujo asociado a la funcionalidad semejanzaAction(Request \$request).

La complejidad ciclomática es la métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa.

Esta métrica calcula la cantidad de caminos independientes de cada una de las funcionalidades del programa. También provee el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez (22).

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad sea el correcto.

1. $V(G) = R$ donde R representa la cantidad total de regiones.

$$V(G) = 3$$

2. $V(G) = A - N + 2$ donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.

$$V(G) = 7 - 6 + 2$$

$$V(G) = 3$$

1. $V(G) = P + 1$ donde P es el número de nodos predicado contenidos en el grafo de flujo (se denomina nodo predicado a los nodos de los cuales parten dos o más aristas).

$$V(G) = 2 + 1$$

$$V(G) = 3$$

Dado a que el cálculo de las tres fórmulas anteriormente mencionadas arrojó el mismo resultado se puede plantear que la complejidad ciclomática del método es 5. Esto significa que existen 3 posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

- **Camino básico #1:** 1-2-4-6.
- **Camino básico #2:** 1-2-3-2-4-6.
- **Camino básico #3:** 1-2-3-2-4-5-4-6.

Para cada camino básico determinado se realiza un diseño de caso de prueba.

- **Caso de prueba para el camino básico # 1:** Si (\$situaciones==" " && \$comparaciones==" ").
- **Caso de prueba para el camino básico # 2:** Si (\$situaciones!=" " && \$comparaciones==" ").
- **Caso de prueba para el camino básico # 3:** Si (\$situaciones!=" " && \$comparaciones!=" ").

Se puede concluir que al aplicar la prueba el resultado arrojado fue que para que se ejecute cada sentencia al menos una vez, el límite superior de casos de pruebas debe ser 3.

Pruebas de caja negra: Para evaluar los requisitos funcionales del sistema de gestión y análisis de las decisiones arquitectónicas del sistema de gestión integral Xedro-ERP se realizaron pruebas de caja negra. Para aplicarla se confeccionaron los diseños de caso de prueba para cada una de las funcionalidades del sistema.

Para comprobar la calidad del sistema fueron realizadas dos iteraciones de pruebas por el grupo de calidad del centro CEIGE. Las no conformidades encontradas se clasifican en:

- No conformidades detectadas en la documentación.
- No conformidades detectadas en la aplicación.

Resultados de las pruebas

En la siguiente tabla se recoge la cantidad de no conformidades detectadas en la documentación y en la aplicación por cada iteración.

Tipo de no conformidades	Documentación	Aplicación
Primera iteración		
Significativa	12	17
No significativa	0	0
Total	12	17
Segunda iteración		
Significativa	0	0
No significativa	0	0
Total	0	0

Tabla 5: Tabla de las no conformidades detectadas en la documentación y en la aplicación por iteraciones.

A continuación se muestra un gráfico con el porcentaje que representan las no conformidades por cada una de las iteraciones.

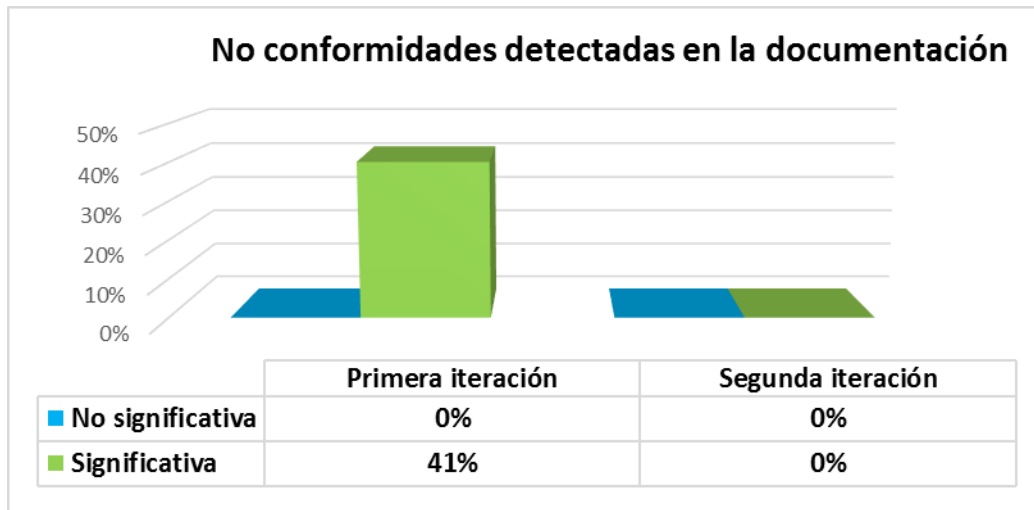


Figura 21. Por ciento que representan las No conformidades detectadas en la documentación para cada una de las iteraciones.

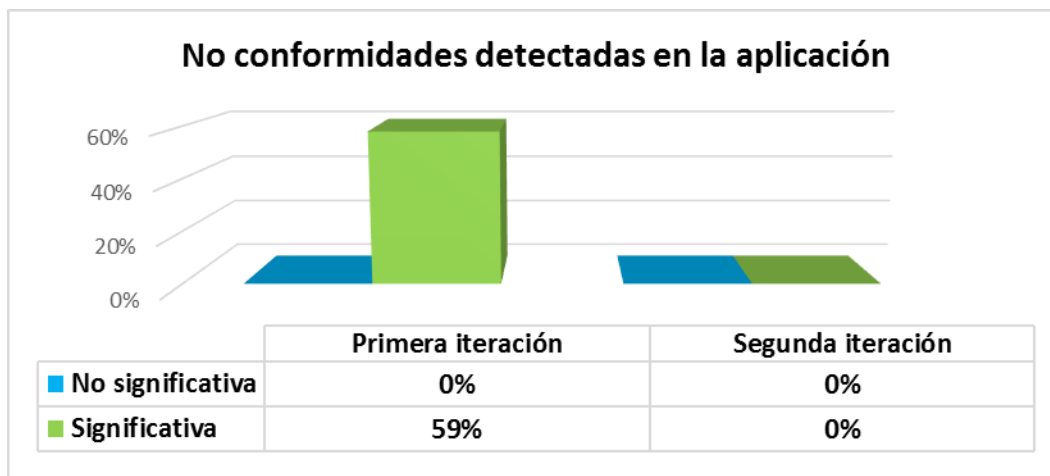


Figura 22. Por ciento que representan las No conformidades detectadas en la aplicación por cada una de las iteraciones.

Pruebas de Aceptación

Otra de las pruebas realizadas fueron las de aceptación con el cliente, que en este caso son los arquitectos principales del CEIGE, donde se evidencia la aprobación del correcto funcionamiento del sistema, debido a que satisface los requisitos funcionales que se identificaron, ver [Anexo 10](#).

3.7 Validación de las variables de investigación

Durante el proceso de investigación en el caso de la variable gestión se valida obteniendo como resultado el sistema que permite gestionar las decisiones arquitectónicas, ejemplo de ello GestionarSituación, GestionarDecisión, encargadas de adicionar, modificar, eliminar, listar y mostrar todo lo referente a estas entidades y en el caso de identificar patrones con la funcionalidad Buscar situaciones semejantes.

3.8 Conclusiones del capítulo

- En el presente capítulo se expusieron los artefactos generados durante la fase de implementación del sistema.
- Se realizaron pruebas de Caja blanca, utilizando la técnica camino básico, permitiendo conocer el mínimo de casos de pruebas a realizar.
- A partir de las dos iteraciones de pruebas funcionales realizadas por el grupo de calidad del CEIGE, se concluye que las funcionalidades implementadas mostraron un correcto funcionamiento y dan respuesta a los requisitos funcionales.
- Una vez concluido el sistema se realizaron las pruebas de aceptación con el cliente, que en este caso fueron los arquitectos y especialistas en el tema de investigación, los cuales arrojaron la aprobación del correcto funcionamiento del producto, debido a que satisface los requisitos funcionales que se identificaron.

Conclusiones Generales

Luego de realizada la investigación para el desarrollo del sistema que gestione y analice las decisiones arquitectónicas en el proceso de desarrollo del Sistema Integral de Gestión Xedro-ERP, se arriba a las siguientes conclusiones:

- Se realizó un estudio del estado del arte sobre el proceso de gestión y análisis de decisiones arquitectónicas, donde se obtuvo como resultado que hasta el momento no hay ningún proceso definido que solucione la situación problemática expuesta en la investigación.
- Se realizó el análisis y diseño del sistema generando los artefactos establecidos por el modelo de desarrollo del CEIGE para dar paso al proceso de implementación.
- Se desarrolló el sistema para la gestión y análisis de las decisiones arquitectónicas en el CEIGE, empleando herramientas, tecnologías libres obteniendo un producto funcional acorde a los requisitos identificados.
- Se empleó Minería de textos en la implementación, haciendo uso de las técnicas análisis semántico latente (LSA) y descomposición de valor singular (SVD), facilitando el desarrollo de la funcionalidad Buscar Situaciones semejantes.
- Al sistema se le realizaron pruebas funcionales, de caja blanca y de aceptación, permitiendo evaluar la validación y verificación del producto.

Por lo anteriormente expuesto se concluye que el sistema gestiona y analiza las decisiones arquitectónicas dándole cumplimiento a los objetivos trazados en la investigación .

Recomendaciones

Se recomienda continuar la investigación y paralelamente definir indicadores a evaluar en el proceso de definir los problemas arquitectónicos, dichos indicadores permitiría conformar una base de conocimientos para aplicar el razonamiento basado en casos como técnica de inteligencia artificial. Además, la aplicación de algoritmos de lógica difusa deviene valor agregado al razonamiento basado en casos, cuando la relevancia de los indicadores no se conoce con entera certeza.

Bibliografía

1. **Portelles, Juan Alberto Caballero.** “Desarrollo de un sistema basado en casos para la gestión de errores en el Proyecto ERP-Cuba” [Tesis]. La Habana : s.n., 2011.
2. IEEE Std 1471-2000. [En línea] 2010. [Citado el: 30 de 12 de 2013.] <http://standards.ieee.org/findstds/standard/1471-2000.html>.
3. **R, Kazman.** FTP Directory.Tool Support for Aechitecture Analysis and Design. [En línea] 1996. [Citado el: 30 de 11 de 2013.] ftp://ftp.sei.cmu.edu/pub/sati/Papers_and_Adstracts/ISAW-2.ps.
4. Definición.de . [En línea] 2008-2013. [Citado el: 30 de 11 de 2013.] <http://definicion.de/toma-de-decisiones/>.
5. Modeling and sharing architectural decisions,Parte 1:Concepts.
6. **Hammer, University of Groningen Department od Mathematic Computing Science.** Tool support for Arquitectural Decisions.
7. **CEIGE, Subdirección de Producción.2013.** Modelo de desarrollo de software. La Habana: Universidad de las Ciencias Informáticas : s.n., 2013.
8. Doctrine Proyect. [En línea] 2014. [Citado el: 30 de 11 de 2013.] <http://www.doctrine-project.org>.
9. Documentación del Servidor HTTP Apache 2.0. [En línea] 2009. [Citado el: 30 de 11 de 2013.] <http://httpd.apache.org/docs/2.0/es/>.
10. [En línea] [Citado el: 2 20, 2014.] https://swa.univie.ac.at/Architectural_Design_Decision_Support_Framework_%28ADvISE%29.
11. **Davide Falessi and Giovanni Cantone, University of Rome, “Tor Vergata” Rick Kazman, Carnegie Mellon University and University of Hawaii Philippe Kruchten, University of British Columbia.**Decision-Making Techniques for Software Architecture Design: A Comparative Survey 9 17, 2011.
12. Definición.de. [En línea] 2008-2014. [Citado el: 11 30, 2013.], <http://es.thefreedictionary.com/taxonom%C3%ADa>.
13. **Pressman.** *Ingeniería del software. Un enfoque práctico.2001.*
14. **Departamento de ciencias de computación e Inteligencia artificial (DECSAI), Universidad de Granada,** [Citado el:03 23,2014.].
15. **Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. 1996.** *Pattern – Oriented Software Architecture. A System of Patterns.* Inglaterra: John Wiley & Sons, 1996.

16. **Tedeschi, Nicolás. 2013.** MSDN. *¿Qué es un Patrón de Diseño?* [En línea] 2014. [Citado el: 25 de Febrero de 2014.] <http://msdn.microsoft.com/es-es/library/bb972240.aspx> .
17. **Larman, Craig. 1999.** *UML y Patrones*. México: Prentice Hall, 1999.
18. **Prieto, Félix. 2009.** *Patrones de diseño*. España: Departamento de Informática. Universidad de Valladolid, 2009.
19. **Yii Framework.** En: *Best MVC Practices* [En línea]. [Citado: 03 de 23 de 2014]. <http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices>.
20. **Potencier, Fabien.**2011. *Symfony Project* [En línea]. [Citado: 03 de 23 de 2014]. <http://www.symfony.com>.
21. **Twig Framework** [En línea]. [Citado: 03 de 24 de 2014]. <http://twig.sensiolabs.org/>.
22. **Pressman, Roger S.** 2005. *Ingeniería de Software. Un enfoque práctico*. Quinta edición. 2005.
23. **Ganesh, Gunda Sai.** 2008. *Requirements Engineering: Elicitation Techniques*. Suecia: Universidad del Este, Departamento de Tecnología, Matemática y Ciencia de la Computación, 2008. PR003.
24. **Sommerville, Ian.** 2005. *Ingeniería de Software*. 7ma. 2005.
25. **ALTOVA. 2013.** ALTOVA. *Diagramas de secuencia UML*. [En línea] 2013. [Citado: 03 de 24 de 2014.] <http://www.altova.com/es/umodel/sequence-diagrams.html>.
26. **Ivar Jacobson, Grady Booch, James Rumbaugh. 1999.** *El Proceso Unificado de Desarrollo del Software*. s. l.: Addison-Wesley Professional, 1999.
27. *Architectural Decision Identification in Architectural Patterns* [En línea]. [Citado: 04 de 10 de 2014]
28. **Olaf Zimmermann¹ Jana Koehler¹ Frank Leymann²,** *Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design*, Universidad Stuttgart, Institute of Architecture of Application Systems Universidad 38, 70569 Stuttgart, Germany, [En línea]. [Citado: 04 de 10 de 2014] frank.leymann@iaas.uni-stuttgart.de .
29. **Olaf Zimmermann, Thomas Gschwind, Jochen Küster¹, Frank Leymann, Nelly Schuster,** *Reusable Architectural Decision Models for Enterprise Application Development*, [En línea]. [Citado: 04 de 10 de 2014] IBM Research GmbH Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland {olz, thg, jku, nes} [@zurich.ibm.com](mailto:nes@zurich.ibm.com), Universidad Stuttgart, Institute of Architecture of Application Systems Universidad 38, 70569 Stuttgart, Germany, frank.leymann@iaas.uni-stuttgart.de, [En línea]. [Citado: 04 de 10 de 2014].
30. **Mohsen Anvaari,** *Reusing Software Architectural Decisions in Electricity Industry*, PhD Candidate at Norwegian University of Science and Technology, [En línea]. [Citado: 04 de 10 de 2014]. mohsena@idi.ntnu.no .

31. **Marcin Nowak, Cesare Pautasso**, Team Situational Awareness and Architectural Decision Making with the Software Architecture Warehouse, Faculty of Informatics, University of Lugano, Switzerland, , [En línea]. [Citado: 04 de 10 de 2014] marcin.nowak@usi.ch, c.pautasso@ieee.org , <http://saw.inf.usi.ch> .
32. **Feldman, Ronen y Sanger, James**. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge: s.n.
33. **Susan T. Dumais**.2005. "Latent Semantic Analysis". *Annual Review of Information Science and Technology*.
34. **Visconti, Marcello**. 2012. *Fundamentos de Ingeniería de Software*. Chile: Universidad Técnica de Federico Santa María, 2012.
35. **Facultad de Informática – UPM**. fermat.usach.cl. [En línea] [Citado el: 04 de 10 de 2014] fermat.usach.cl/~msanchez/comprimido/OBJETOS.pdf.
36. Tomado del sitio oficial de Geany. [En línea] [Citado el: 05 de 24 de 2014]<http://www.geany.org>.
37. **Bikha**. Requirements Management – Defining the Project Scope and Developing Use Cases. 2008.
38. **Padilla, A.V**. 2010. Sistema Experto para la interpretación mamográfica. México, D.F. [Consultado el: 4 de marzo del 2011] Disponible en: www.paginaspersonales.unam.mx/files/18/SEIM_TesisAVPD.pdf.

Anexos

Anexo 1. Encuesta aplicada a un grupo de arquitectos y especialistas en el tema de investigación del CEIGE.



UCI Universidad de las Ciencias Informáticas

CENTRO DE INFORMATIZACIÓN DE GESTIÓN DE ENTIDADES
Facultad 3

Encuesta a Arquitectos del CEIGE sobre Decisiones Arquitectónicas


Nombre y Apellidos:

Proyecto al que pertenece:

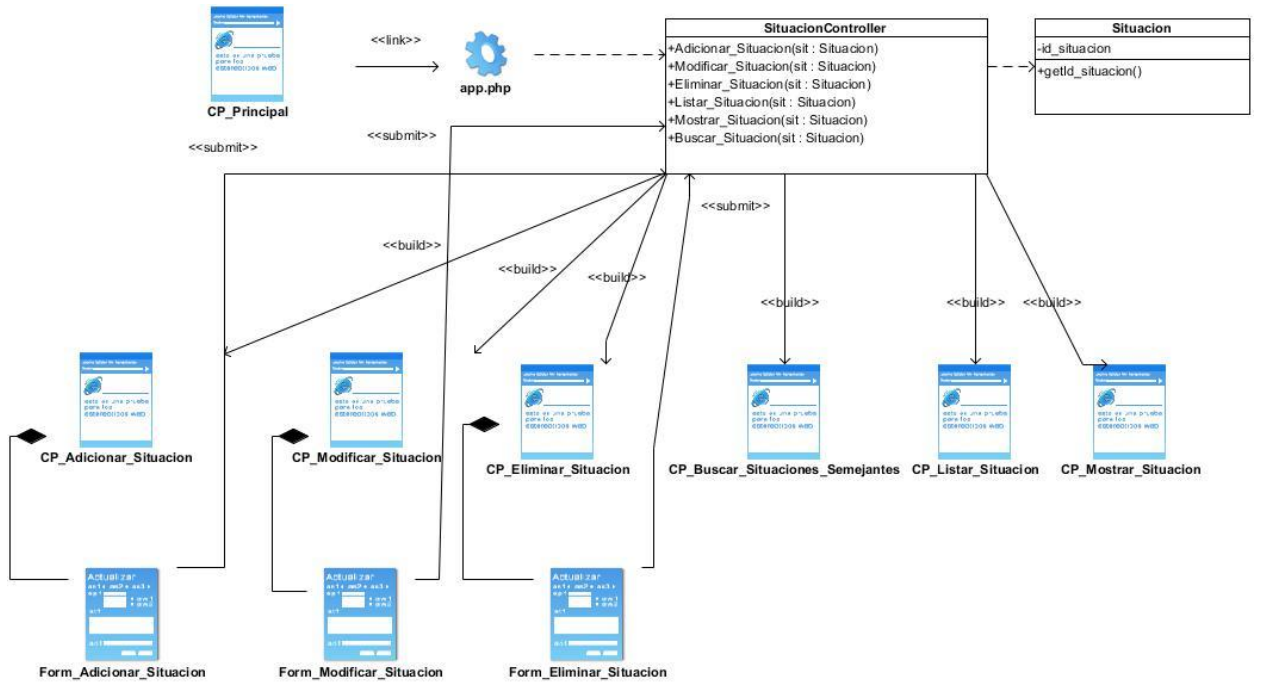
Años de experiencia en el centro:

No	Preguntas	Respuestas
1.	¿Cuándo se presenta un problema arquitectónico en su proyecto es capaz de darle una solución al momento o tienen que consultar otros medios? En caso de que consulte otros medios especifique cuáles.	
2.	¿Cuándo se le da solución a un problema arquitectónico en su proyecto, queda evidencia de la solución tomada? Ha utilizado este registro de evidencia en otros problemas presentados? En caso de registrarlo, especifique dónde.	
3.	Mencione algunas de las decisiones arquitectónicas que se hayan tomado en su proyecto.	

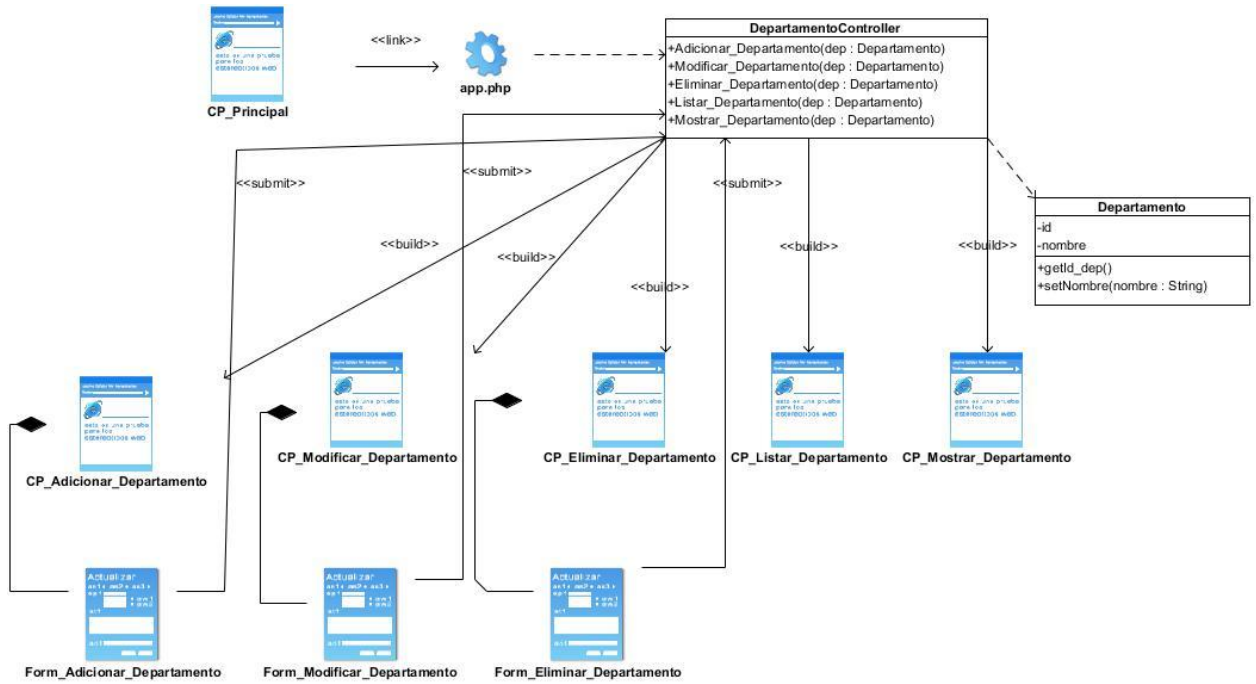
4.	<p>En caso de que en su proyecto se registren las decisiones arquitectónicas, responda:</p> <p>¿Le es engorroso para darle solución a un problema revisar los documentos emitidos con las decisiones arquitectónicas que se han tomado para el problema en cuestión?</p>	
5.	<p>¿Aplica algún mecanismo para reutilizar decisiones arquitectónicas que se toman en otros proyectos?</p>	
6.	<p>¿Cree que le sería más factible a la hora de tomar una decisión arquitectónica en su proyecto, el contar con una aplicación que recoja decisiones que se hayan tomado anteriormente ante un problema semejante al identificado por usted? Explique.</p>	
7.	<p>¿Cuáles de los siguientes atributos utiliza en su proyecto o a su criterio son relevantes a la hora de registrar una decisión arquitectónica?</p>	<p>___ problema ___ decisión ___ estatus de la decisión ___ pendientes ___ decidido ___ aprobado ___ grupo (integración, presentación o datos)</p>

 Universidad de las Ciencias Informáticas		CENTRO DE INFORMATIZACIÓN DE GESTIÓN DE ENTIDADES Facultad 3
		___ suposiciones(decisión-costos, cronograma, tecnología) ___ limitaciones ___ posiciones(opciones o alternativas viables) ___ argumentos ___ implicaciones ___ decisiones relacionadas ___ artefactos relacionados ___ principios relacionados ___ notas
8.	¿Cómo almacenan las decisiones arquitectónicas?	

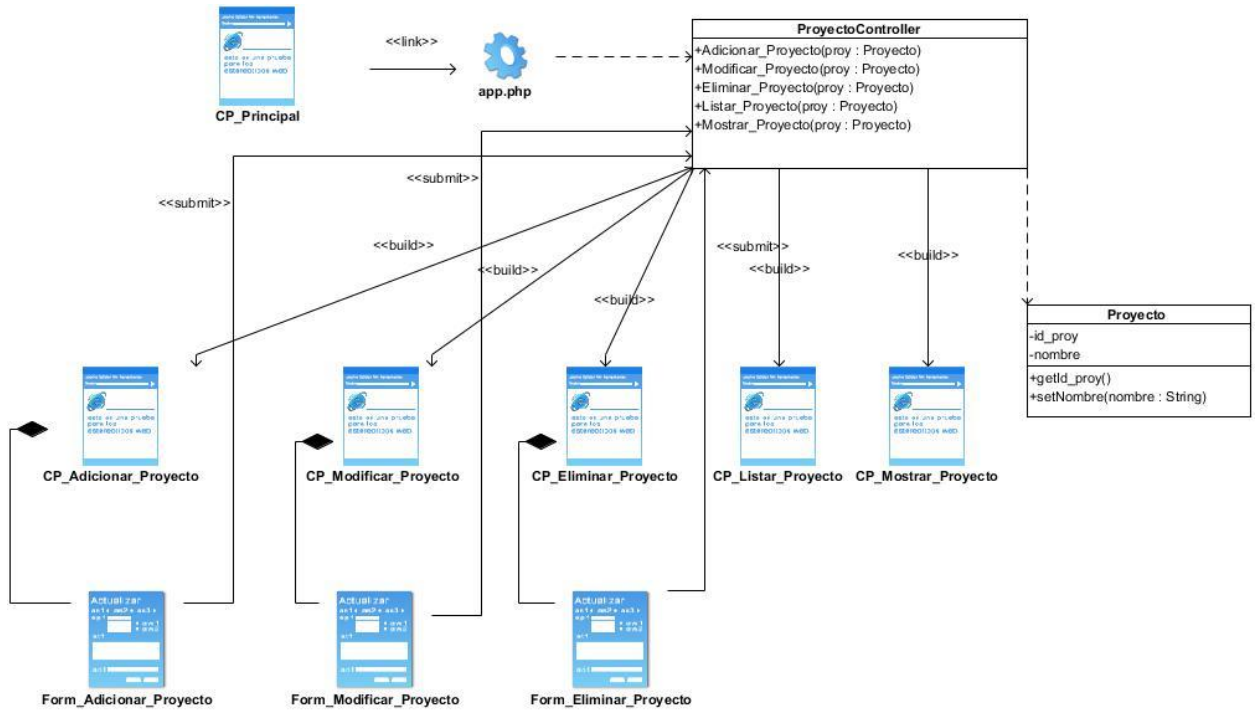
Anexo 2. Diagramas de clases del diseño perteneciente al requisito **Gestionar Situación**.



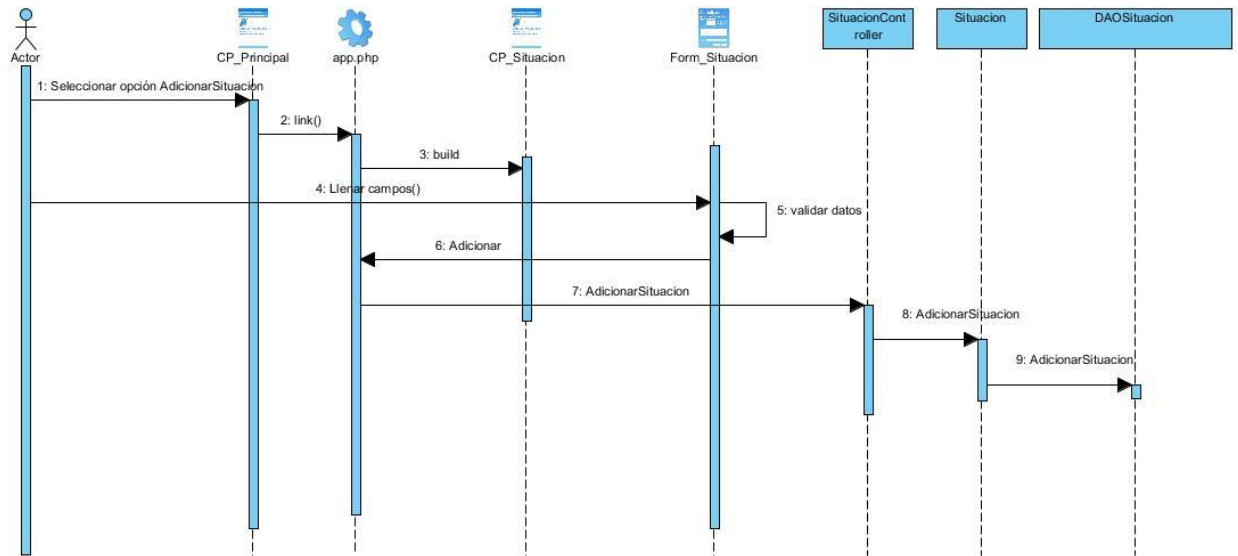
Anexo 3. Diagramas de clases del diseño perteneciente al requisito Gestionar Departamento.



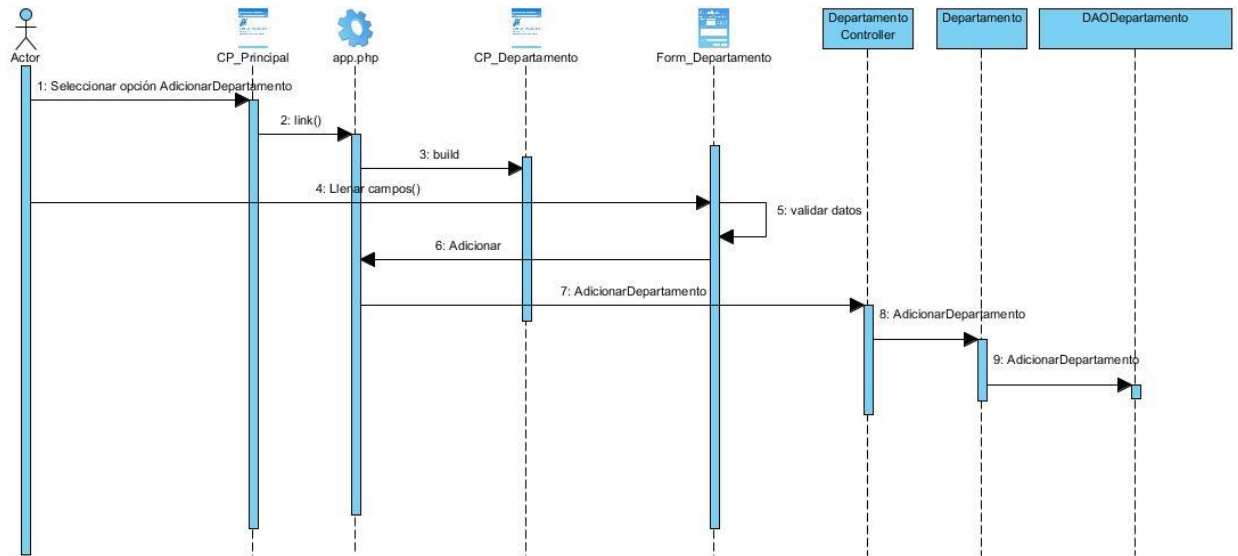
Anexo 4. Diagramas de clases del diseño perteneciente al requisito **Gestionar Proyecto**.



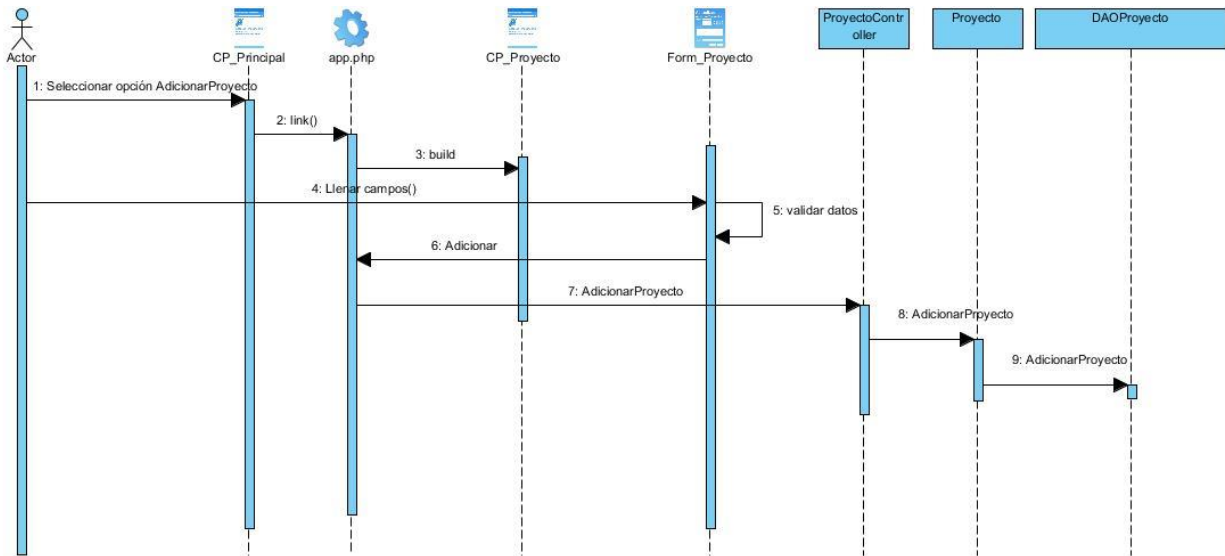
Anexo 5. Diagrama de secuencia perteneciente al requisito **Adicionar Situación**.



Anexo 6. Diagrama de secuencia perteneciente al requisito Adicionar Departamento.



Anexo 7. Diagrama de secuencia perteneciente al requisito Adicionar Proyecto.



Anexo 8. Resultados de la aplicación de la métrica TOC para una muestra de clases del sistema.

No	Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
1	adUser	20	Alta	Alta	Baja
2	adUserProvider	10	Media	Media	Media
3	DefaultController	1	Baja	Baja	Alta
4	DepartamentoController	11	Media	Media	Media
5	NotificacionController	3	Baja	Baja	Alta
6	ProyectoController	11	Media	Media	Media
7	TesisAdministracionExtension	1	Baja	Baja	Baja
8	Departamento	8	Media	Media	Media
9	Proyecto	17	Alta	Alta	Baja
10	DepartamentoType	3	Baja	Baja	Alta
11	ProyectoType	3	Baja	Baja	Alta
12	RolController	11	Media	Media	Media
13	UsuarioController	12	Media	Media	Media
14	TesisSeguridadExtension	1	Baja	Baja	Alta
15	Rol	8	Media	Media	Media
16	Usuario	12	Media	Media	Media
17	RolType	3	Baja	Baja	Alta
18	UsuarioType	3	Baja	Baja	Alta
29	LoadActividades	2	Baja	Baja	Alta
20	LoadDepartamentos	2	Baja	Baja	Alta
21	LoadDisciplinas	2	Baja	Baja	Alta
22	LoadEstados	2	Baja	Baja	Alta
23	LoadFases	2	Baja	Baja	Alta
24	LoadInicial	1	Baja	Baja	Alta
25	LoadProyectos	2	Baja	Baja	Alta
26	LoadRoles	2	Baja	Baja	Alta
27	LoadUsuarios	2	Baja	Baja	Alta
28	TesisConfiguracionExtension	1	Baja	Baja	Alta
29	Articulo	3	Baja	Baja	Alta
30	Caracterespecial	3	Baja	Baja	Alta
31	Conjuncion	3	Baja	Baja	Alta
32	Contraccion	3	Baja	Baja	Alta
33	Disyuncion	3	Baja	Baja	Alta
34	Preposicion	3	Baja	Baja	Alta
35	ActividadController	11	Media	Media	Media
36	ArtefactoController	11	Media	Media	Media
37	DecisionController	14	Media	Media	Media

38	DisciplinaController	11	Media	Media	Media
39	EstadoController	10	Media	Media	Media
40	FaseController	11	Media	Media	Media
41	SituacionController	34	Alta	Alta	Baja
42	WelcomeController	1	Baja	Baja	Alta
43	TesisPublicoExtension	1	Baja	Baja	Alta
44	Actividad	15	Alta	Alta	Baja
45	Artefacto	11	Media	Media	Media
46	Decision	32	Alta	Alta	Baja
47	Disciplina	12	Media	Media	Media
48	Estado	8	Media	Media	Media
49	Fase	11	Media	Media	Media
50	Situacion	18	Alta	Alta	Baja
51	ActividadType	3	Baja	Baja	Alta
52	ArtefactoType	3	Baja	Baja	Alta
53	DecisiType	3	Baja	Baja	Alta
54	DecisioType	4	Baja	Baja	Alta
55	DecisionType	3	Baja	Baja	Alta
56	DisciplinaType	3	Baja	Baja	Alta
57	EstadoType	3	Baja	Baja	Alta
58	FaseType	3	Baja	Baja	Alta
59	SituacionType	3	Baja	Baja	Alta

Anexo 9. Resultados de la aplicación de la métrica RC para una muestra de clases del sistema.

No	Clase	Cantidad de relaciones	Acoplamiento	Complejidad de mantenimiento	Reutilización	Cantidad de pruebas
1	adUser	0	Ninguna	Baja	Alta	Baja
2	adUserProvider	0	Ninguna	Baja	Alta	Baja
3	DefaultController	0	Ninguna	Baja	Alta	Baja
4	DepartamentoController	2	Media	Media	Media	Media
5	NotificacionController	0	Ninguna	Baja	Alta	Baja
6	ProyectoController	2	Media	Media	Media	Media
7	TesisAdministracionExtension	0	Ninguna	Baja	Alta	Baja
8	Departamento	1	Baja	Baja	Alta	Baja
9	Proyecto	3	Alta	Alta	Baja	Alta
10	DepartamentoType	0	Ninguna	Baja	Alta	Baja
11	ProyectoType	0	Ninguna	Baja	Alta	Baja
12	RolController	2	Media	Media	Media	Media
13	UsuarioController	2	Media	Media	Media	Media
14	TesisSeguridadExtension	0	Ninguna	Baja	Alta	Baja
15	Rol	1	Baja	Baja	Alta	Baja
16	Usuario	2	Media	Media	Media	Media
17	RolType	0	Ninguna	Baja	Alta	Baja
18	UsuarioType	0	Ninguna	Baja	Alta	Baja
29	LoadActividades	1	Baja	Baja	Alta	Baja
20	LoadDepartamentos	1	Baja	Baja	Alta	Baja
21	LoadDisciplinas	1	Baja	Baja	Alta	Baja
22	LoadEstados	1	Baja	Baja	Alta	Baja
23	LoadFases	1	Baja	Baja	Alta	Baja
24	LoadInicial	5	Alta	Alta	Baja	Alta
25	LoadProyectos	1	Baja	Baja	Alta	Baja
26	LoadRoles	1	Baja	Baja	Alta	Baja
27	LoadUsuarios	1	Baja	Baja	Alta	Baja
28	TesisConfiguracionExtension	0	Ninguna	Baja	Alta	Baja
29	Articulo	0	Ninguna	Baja	Alta	Baja
30	Caracterespecial	0	Ninguna	Baja	Alta	Baja
31	Conjuncion	0	Ninguna	Baja	Alta	Baja
32	Contraccion	0	Ninguna	Baja	Alta	Baja
33	Disyuncion	0	Ninguna	Baja	Alta	Baja
34	Preposicion	0	Ninguna	Baja	Alta	Baja

35	ActividadController	2	Media	Media	Media	Media
36	ArtefactoController	2	Media	Media	Media	Media
37	DecisionController	4	Alta	Alta	Baja	Alta
38	DisciplinaController	2	Media	Media	Media	Media
39	EstadoController	2	Media	Media	Media	Media
40	FaseController	2	Media	Media	Media	Media
41	SituacionController	2	Media	Media	Media	Media
42	WelcomeController	0	Ninguna	Baja	Alta	Baja
43	TesisPublicoExtension	0	Ninguna	Baja	Alta	Baja
44	Actividad	4	Alta	Alta	Baja	Alta
45	Artefacto	2	Media	Media	Media	Media
46	Decision	4	Alta	Alta	Baja	Alta
47	Disciplina	2	Media	Media	Media	Media
48	Estado	1	Baja	Baja	Alta	Baja
49	Fase	2	Media	Media	Media	Media
50	Situacion	3	Alta	Alta	Baja	Alta
51	ActividadType	0	Ninguna	Baja	Alta	Baja
52	ArtefactoType	0	Ninguna	Baja	Alta	Baja
53	DecisiType	0	Ninguna	Baja	Alta	Baja
54	DecisioType	0	Ninguna	Baja	Alta	Baja
55	DecisionType	0	Ninguna	Baja	Alta	Baja
56	DisciplinaType	0	Ninguna	Baja	Alta	Baja
57	EstadoType	0	Ninguna	Baja	Alta	Baja
58	FaseType	0	Ninguna	Baja	Alta	Baja
59	SituacionType	0	Ninguna	Baja	Alta	Baja

Anexo 10. Acta de Conformidad del cliente.



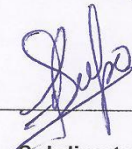
Acta de aceptación

CENTRO DE INFORMATIZACIÓN DE GESTIÓN DE ENTIDADES

26 de May de 2014

En cumplimiento de el *Sistema de gestión de las decisiones arquitectónicas* y en función de su ejecución en el Centro de Informatización de Gestión de Entidades (CEIGE), se hace entrega del producto aprobando el correcto funcionamiento del mismo por el cliente, debido a que satisface los requisitos funcionales que se identificaron.

Entrega: Sistema de gestión de las decisiones arquitectónicas	Recibe: Subdirector de producción	
Nombre y Apellidos: Lisvany Hernández Cabrera Elizabeth Pérez Miranda	Nombre y Apellidos: Ing. Dailenys de la Caridad Arias Pupo	Cargo: Subdirector de producción
Nombre y apellidos de los tutores: Ing. Dailenys de la Caridad Arias Pupo Ing. Gisselle Almedia González Ing. Karel Riverón Escobar	Certifica: Ing. Ariadna Rendón Artola (Arquitecto de sistema de BK Import/Export) Ing. William González Obregón (Arquitecto principal del CEIGE)	


 Subdirector de producción
 Ing. Dailenys de la Caridad Arias Pupo

Anexo 11. Interfaz de la funcionalidad Situaciones semejantes.



The screenshot shows the SIGELDA web interface. On the left is a dark sidebar with a teal header containing the 'SIGELDA' logo and a menu icon. Below the header are three menu items: 'Funcionalidades', 'Administración', and 'Público'. The main content area has a teal header with the text 'SITUACIONES SEMEJANTES A: PROBLEMAS CON LA CONFIGURACIÓN DE LOS DATOS'. Below this is a table with three columns: 'Problema', 'Semejanza', and 'Acción'. The table contains two rows of data. The first row shows a problem with 100% similarity and an eye icon. The second row shows a problem with 50% similarity and an eye icon. At the bottom of the main area are three buttons: 'Exportar', 'Enviar', and 'Atrás'.

	Problema	Semejanza	Acción
1	problemas con la configuración de los datos	100%	
2	mala gestión de datos	50%	

Glosario

A.

Arquitectura de software: Es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución.

M.

Métrica: Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

S.

Sistema: Es un conjunto organizado de objetos o partes interactuantes e interdependientes, que se relacionan formando un todo unitario y complejo.

Software: conjunto de instrucciones que los ordenadores emplean para manipular datos.

V.

Validación: Confirmación mediante el suministro de evidencia objetiva de que se han cumplido los requisitos para una utilización o aplicación específica prevista.