Universidad de las Ciencias Informáticas

Facultad 8

"Diseño e implementación del módulo de Gestión de indicadores del proyecto del Ministerio del Poder Popular de la Comunicación y la Información"

Autores: Abelardo López Quintana.

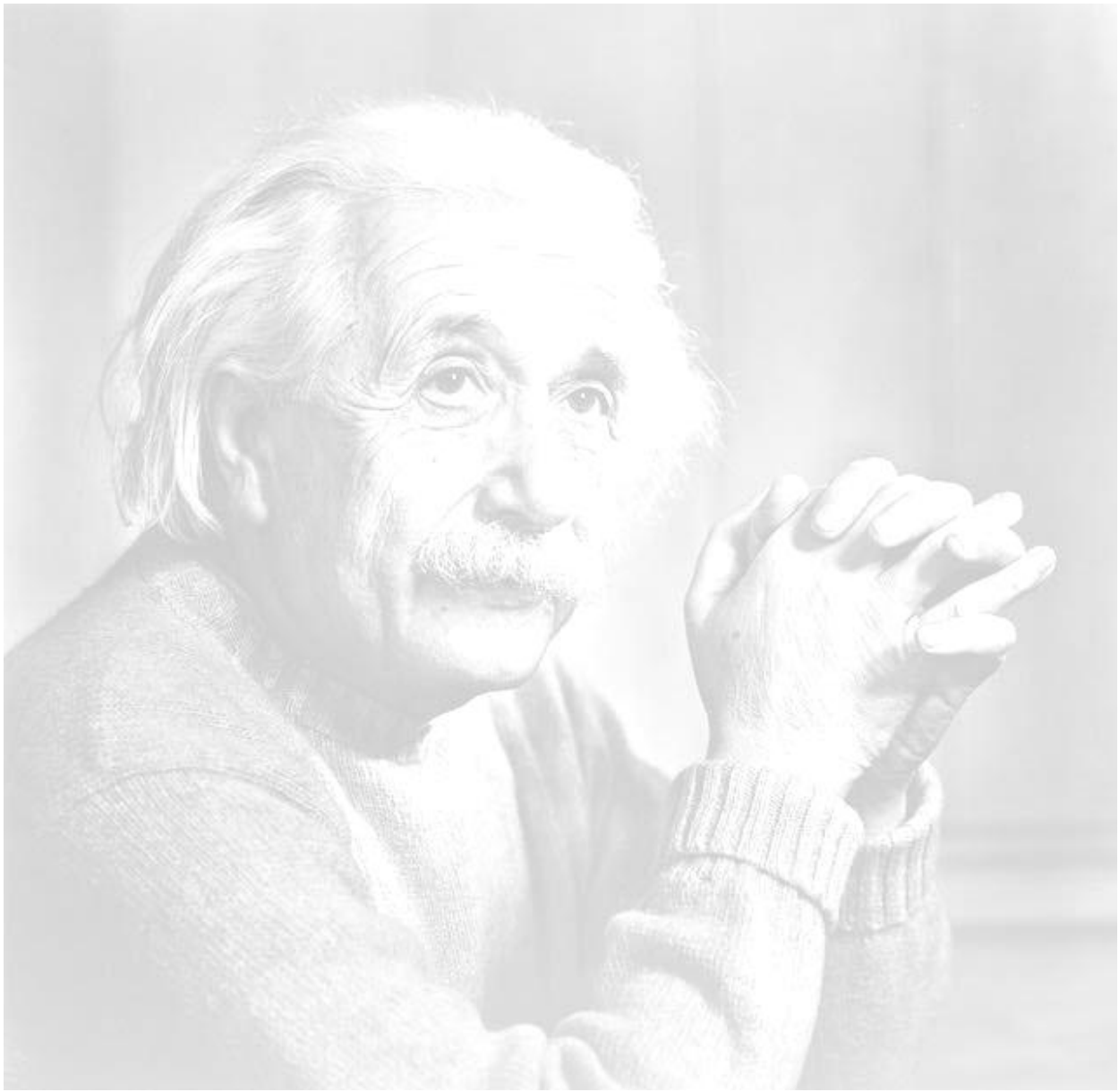
Raúl Romero Leyva .

Tutores: Ing. Yuniesky Armentero Moreno

Ing. Dismey Saavedra López

Junio 2010

¨Año 52 Aniversario del Triunfo de la Revolución¨



“La imaginación es más importante que el conocimiento. El conocimiento es limitado, mientras que la imaginación no”

Albert Einstein

Declaración de autoría

Declaramos que somos los únicos autores del trabajo “Diseño e implementación del módulo de Gestión de indicadores del proyecto del Ministerio del Poder Popular de la Comunicación y la Información” y autorizamos a la Facultad 8 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autores:

_____

Abelardo López Quintana

_____

Raúl Romero Leyva

Tutores:

_____

Ing. Yuniesky Armentero Moreno

_____

Ing. Dismey Saavedra López

Agradecimientos

Abelardo López Quintana.

A todas las personas que me ayudaron en la realización de este trabajo, en especial a mi novia Mailen que ha estado a mi lado durante estos cinco años.

A mis tutores Yuniesky y Dismey quienes fueron los que nos guiaron siempre.

A todos mis amigos que estuvieron siempre a mi lado.

A mis padres Abelardo y Gisela que siempre me apoyaron en mis decisiones y fueron los principales guías en este gran logro.

A mi familia en general, a mi hermano Abel que me ayudó mucho en seguir en la escuela.

A mi tía Gloria y a mi primo Raulito por su incondicionalidad mostrada en todo momento.

A mi compañero de tesis Raúl por estar a mi lado en esta difícil tarea.

A mi tribunal de tesis y mi oponente que siempre me refinaron y arreglaron errores en el trabajo de diploma.

A la Revolución por darme la oportunidad de realizar mi sueño.

Raúl Romero Leyva.

A mis padres Addys y Raúl por apoyarme en todo momento y confiar en mí, a mi novia Ayarisé por estar a mi lado en los momentos difíciles.

A mi familia en especial a mis tíos Idania y Juan Carlos y a mis primos Juancarlitos y Denia por acogerme siempre que los molestaba.

A mi compañero de tesis Abelardo sin él no hubiera sido posible este logro.

A mi tutor Yuniesky y Dismey por guiarnos y ayudarnos en la realización de este trabajo.

A mis amigos del apto, del grupo y de la escuela.

En general todas aquellas personas que de una forma u otra han ayudado en la realización de este trabajo.

A la Revolución por permitirme estudiar en esta escuela y serle útil a la sociedad.

Dedicatoria

Abelardo López Quintana.

A mi mamá y mi papa por todo su cariño y apoyo, por haberme guiado siempre en todos mis estudios y por ser a quien le debo todos mis resultados, sin su guía nunca hubiera llegado a este momento.

A toda mi inmensa familia por su respeto y honestidad mostrada durante estos cinco años.

A mis profesores por formarme como profesional.

A mi novia por soportarme durante estos cinco años de carrera.

A la universidad por ser mi gran casa y ser el lugar donde me formé.

A mis abuelos por ser las personas que me inspiraron a ser ingeniero.

Raúl Romero Leyva

A mis padres por todo su amor y apoyo incondicional, por ser siempre los guías y por indicarme el camino correcto a seguir.

A toda mi inmensa familia que de una forma u otra me apoyaron desde la distancia durante estos cinco años.

A mi novia, a mis profesores y a la universidad.

A mis amigos.

Resumen

Actualmente Cuba está llevando a cabo convenios con diversos países en muchas de las esferas de la sociedad, una de ellas es la informática, es por esto que el Ministerio del Poder Popular de la Comunicación y la Información (MINCI) de la República Bolivariana de Venezuela y la Universidad de las Ciencias Informáticas (UCI) desarrollan un proyecto en conjunto, el cual permitirá informatizar la Gestión de indicadores de proyectos.

Este trabajo se basa en las actividades realizadas por los autores en los roles de diseñador e implementador como integrantes del proyecto DATEC. En él se muestra el diseño y la implementación de la solución brindada para el módulo de Gestión de indicadores, a partir de los requisitos definidos anteriormente por los analistas.

Para darle cumplimiento a las funcionalidades esperadas por el módulo, se realizó un diseño basado en patrones y para su implementación, se utilizaron herramientas y tecnologías de la plataforma Zend que exponen tendencias de la programación Web y que en su mayoría son libres y de código abierto.

Índice

Índice de figuras

Índice de tablas

Introducción

El uso de las Tecnologías de la Información y las Comunicaciones (TIC) ha aumentado considerablemente en la actualidad, siendo la principal protagonista del desarrollo continuo del software.

Hoy en día los productos informáticos son desarrollados principalmente por las grandes potencias mundiales, trayendo como consecuencia la dependencia del software, al ser estos productos de un alto valor en el mercado mundial. Es por esto que Cuba, país del tercer mundo y con pocos recursos naturales, se ha envuelto en el desarrollo de esta rama, dirigido principalmente a informatizar el país y la cooperación con diferentes países, principalmente con la República Bolivariana de Venezuela, con la cual se desarrollan un conjunto de proyectos encaminados a la mejora de la sociedad venezolana.

La Universidad de la Ciencias Informáticas (UCI) es una estrategia trazada con el objetivo de llevar la informática a todos los rincones del país. Esta institución tiene como misión insertar a Cuba en el mercado mundial del software, aprovechando así la capacidad intelectual e inventiva de los cubanos. En ella se pueden encontrar diferentes centros productivos, destacándose el Centro de Tecnologías de Análisis de Datos (DATEC), el mismo resuelve la dependencia de otros países debido a que se implementan software libres de fácil uso, además se apoya en la gestión y desarrollo de productos informáticos.

La mayoría de estos productos son desarrollados para todos los países de la región y del mundo, siendo Venezuela el país que más convenio tiene con la universidad, ya que se aspira a que todas sus instituciones usen productos desarrollados en software libre y que los mismos permitan agilizar los procesos de informatización de la sociedad.

En este contexto surge el proyecto del Ministerio del Poder Popular de la Comunicación y la Información (MINCI), el cual tendrá como principal característica regir la política de comunicación, información y publicidad de la gestión del gobierno nacional, con el fin de avanzar en los principios establecidos en la Constitución Nacional de la República Bolivariana de Venezuela y así fomentar la utilización de los medios de comunicación como instrumento de formación, logrando mayor participación ciudadana en la gestión pública y en el control eficiente de los indicadores de gestión para un seguimiento y evaluación del acceso a los diferentes tipos de información, además de una correcta planificación estratégica y la gestión de

proyectos del Ministerio, el control de indicadores de gestión y su impacto de cada uno de los proyectos.

Por este motivo se necesita una herramienta informática capaz de automatizar los objetivos del Ministerio, la gestión de proyectos y el control de indicadores de gestión.

De tal situación problémica surge el siguiente **problema a resolver:** la necesidad de gestionar indicadores de proyecto para el MINCI. El problema planteado se enmarca en el **objeto de estudio:** sistemas de Gestión de indicadores, delimitado por el **campo de acción:** diseño e implementación de sistemas de Gestión de indicadores.

Para resolver el problema planteado se ha propuesto como **objetivo general:** diseñar e implementar el módulo de Gestión de indicadores del proyecto del MINCI.

Para alcanzar el mismo se trazan los siguientes **objetivos específicos:**

1. Realizar un estudio del estado del arte sobre los patrones de diseño y buenas prácticas de programación.
   a. Realizar una evaluación del diseño del marco de trabajo Cedrux.
2. Diseñar las clases del módulo de Gestión de indicadores del proyecto MINCI.
3. Implementar el módulo de Gestión de indicadores del proyecto del MINCI.

Como **idea a defender:** se tiene que si se desarrolla un producto informático para la Gestión de indicadores con las particularidades del proyecto, entonces se logrará una mejor planificación para el Ministerio; logrando con esto, un buen control en la gestión pública y en los recursos con los que dispone el mismo para la ejecución de los proyectos.

Posibles resultados:

✓ Desarrollar un módulo de Gestión de indicadores que permita fomentar las TIC en la población.

✓ Promover el control social de la población hacia los medios de comunicación masivos.

✓ Alinear la ejecución de los proyectos con los objetivos estratégicos del Ministerio.

✓ Aumentar el control sobre los recursos que dispone el Ministerio para la ejecución de los proyectos.

✓ Dotar al MINCI de un sistema informático integral que permita mejorar la eficacia de su funcionamiento.

Métodos de investigación científica:

Los métodos de investigación a emplear tienen como punto central la interacción dialéctica de los métodos teóricos y empíricos.

Los métodos teóricos a utilizar son los siguientes:

✓ **Análisis histórico-lógico**: permitió un estudio del estado del arte haciendo un análisis de los fenómenos relacionados con los sistemas informáticos implementados en el país y en el mundo, así como sus ventajas y desventajas. Además de un estudio de la gestión de indicadores.

✓ **Analítico-Sintético:** permitió buscar la esencia de los fenómenos, en este caso de los sistemas informáticos implementados en Cuba y el mundo, los rasgos que los caracterizan y los distinguen.

El método empírico a utilizar es el siguiente:

Observación: con el objetivo de obtener información, datos y otros aspectos con los que no se contaba para el desarrollo de la tesis.

Estructura capitular:

Capítulo I: Fundamentación teórica. Se abordarán todos los elementos teóricos que sustentan el problema científico y los objetivos del trabajo. Además de un análisis de las diferentes herramientas, metodologías y tecnologías usadas.

Capítulo II: Diseño propuesto. Se analizarán los elementos fundamentales a tener en cuenta en el desarrollo del producto, realizando una valoración crítica del diseño para el módulo. Principalmente los diagramas de clases e interacción.

Capítulo III: Implementación y Pruebas. Quedarán establecidas las interfaces del sistema de gestión para el módulo de indicadores, así como la validación de la solución propuesta, la implementación del producto, además se realizarán las pruebas unitarias al producto y las métricas de diseño.

Para desarrollar el presente trabajo se llevarán a cabo las siguientes **tareas de investigación:**

✓ Estudiar y analizar los patrones de diseño.

✓ Estudiar los sistemas de Gestión de indicadores.

✓ Diseñar el módulo de Gestión de indicadores.

✓ Implementar el módulo de Gestión de indicadores en cada una de sus iteraciones.

✓ Realizar las diferentes pruebas unitarias del producto siguiendo un método formal de evolución.

Capítulo1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se abordará sobre las diferentes herramientas a utilizar en el módulo. Se describirán los Frameworks más usados, así como su importancia y uso en el mundo, además se caracterizarán algunos IDEs utilizados en el desarrollo del sistema de gestión que sirven para hacer interfaces Web amigables, rápidas y funcionales.

Otro aspecto a tratar, es todo lo referente a cómo se caracterizan las tecnologías y metodologías usadas en el desarrollo del módulo, las cuales fueron escogidas por su importancia en el desarrollo de software. Además, se realizará un estudio referencial del estado actual de los sistemas de gestión, específicamente la Gestión de indicadores.

1.2 Sistemas relacionados con la gestión de indicadores

En la actualidad se desarrollan un gran número de sistemas que sirven para que el proceso de producción se lleve a cabo con eficiencia. Los sistemas de gestión, tanto de proyectos como de indicadores, se empezaron a desarrollar a partir de la década del 90, sirviendo de soporte para la realización de una administración eficiente, brindando soluciones prácticas e integrales a problemas reales. Son sistemas estructurados que buscan satisfacer las demandas de soluciones de gestión empresarial, basados en el concepto de una solución completa que permita a las empresas unificar las diferentes áreas de productividad de la misma.

1.2.1 DotProject

DotProject es una herramienta basada en la Web de gestión de proyectos así como en PHP. Esta potente herramienta utiliza las bases de datos MySQL, teniendo como característica principal su excelente distribución y diseño, permitiendo que las páginas se carguen rápidamente, además de que posee líneas limpias y concisas para su uso. (1)

Su principal desventaja es que su interfaz es un poco compleja, lo que dificulta su uso para la creación y modificación de tareas en la gestión de proyectos.

1.2.2 PHProjekt

Se basa en un sistema de módulo para añadir proyectos o tareas específicas. Este sistema permite al cliente diseñar el proyecto y a los consultores, agregar tareas, plazos y dar informes sobre la marcha, permitiendo al cliente ver exactamente lo que está pasando. PHProjekt está basado en un sistema de PHP y MySQL, y como tal, es un programa Web. (1)

El mismo no posee una fácil adaptación en las diferentes empresas por ser un software con una difícil configuración y no ser muy amigable.

1.2.3 Project Open

Project Open es una completa herramienta de gestión de proyectos, totalmente Web, con importantes funcionalidades tales como: en ella se realiza un seguimiento del flujo de trabajo y un control completo de cada una de las tareas, actividades que se realizan a lo largo del proyecto y dispone de módulos que permiten conectar Project Open con otros sistemas. (1)

Esta herramienta no soporta todos los procesos del área de Gestión de Alcance definidos por el PMBOK.

1.2.4 DotProject en Cuba

En Cuba se implementan diferentes productos informáticos, destinados al cumplimiento de convenios con otros países, principalmente para Venezuela, siendo la UCI la que ha llevado el paso al frente en el desarrollo de estos sistemas de gestión. Uno de los principales proyectos es el Unicornio, que a partir del DotProject, se está adaptando a las necesidades de la universidad.

1.2.5 Planificación de Recursos Empresariales (ERP) en la UCI

El proyecto ERP se dedica a la gestión de proyecto, donde se implementan módulos destinados al control de diferentes actividades, objetivos, o tareas, en dependencia de la empresa u organización a la cual será destinado.

1.3 Metodología de desarrollo

Las metodologías de desarrollo de software describen los pasos para desarrollar un producto de software. Definen detalladamente qué es lo que se debe hacer, cómo y quién debe hacerlo. Precisan cuáles son las tareas a desarrollar durante el ciclo de vida del proyecto, los artefactos que han de ser construidos, cómo deben hacerse y el

orden en que serán realizados, además de designar responsables por cada tarea. De esta forma, se obtiene como producto final, una solución eficaz, factible y de calidad al problema que le dio origen. La selección de las metodologías a utilizar se hace en base a las necesidades específicas de cada situación y a las características del equipo de desarrollo.

La metodología a utilizar es la creada por el Centro Almacenamiento y Gestión de Datos (DATEC) basada en OpenUP.

1.3.1 Metodología para el desarrollo de productos basada en modelo de producción de líneas de productos de software

En ella se definen las fases, actividades y artefactos que se deben generar durante el ciclo de vida del software. Está basada en la adaptación de metodologías para el desarrollo de software en equipo, como son: SCRUM y OpenUP. Se centra en la organización propuesta para la aplicación del Modelo de Líneas de Productos de Software en DATEC. Actualmente se aplica en las líneas de Herramientas y Soluciones Integrales del Centro de Tecnologías de Análisis y Almacenamiento de Datos. (2)

Está estructurada en 4 fases:

- ✓ Inicio
- ✓ Elaboración
- ✓ Construcción
- ✓ Transición

El ciclo de vida para el desarrollo del proyecto está dividido en 9 disciplinas:

- ✓ Estudio preliminar
- ✓ Modelamiento del negocio
- ✓ Requisitos
- ✓ Análisis y diseño
- ✓ Implementación
- ✓ Pruebas internas
- ✓ Pruebas de liberación
- ✓ Despliegue
- ✓ Soporte

1.4 Patrones de diseño de software

Los patrones de diseño de software son una solución probada para un problema general de diseño, en un contexto determinado. Encierran la experiencia de los programadores e ingenieros que han ido adquiriendo en las soluciones de problemas comunes.

Existen diferentes categorías de patrones, los más utilizados en el diseño de software, son los que se caracterizan a continuación.

1.4.1 Patrones GRASP

Patrones Generales de Software para Asignación de Responsabilidades (General Responsibility Assignment Software Patterns). Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Entre las responsabilidades más importantes están:

Conocer:

- ✓ Conocer los datos privados encapsulados.
- ✓ Conocer los objetos relacionados.
- ✓ Conocer las cosas que puede derivar o calcular.

Hacer:

- ✓ Hacer algo él mismo, como crear un objeto o hacer un cálculo.
- ✓ Iniciar una acción en otros objetos.
- ✓ Controlar y coordinar actividades en otros objetos.

1.4.1.1 Bajo Acoplamiento

Su principal característica es mantener las clases más independientes entre sí y con la menor cantidad de relaciones; la cual posibilita que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases, asignándoles una responsabilidad para mantener un bajo acoplamiento. (3)

Ventajas:

- ✓ No se afectan por cambios de otros componentes.
- ✓ Fácil de entender por separado.
- ✓ Fácil de reutilizar.

1.4.1.2 Alta Cohesión

La principal característica de este patrón es asignar responsabilidades de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase. (3)

Ventajas:

- ✓ Mejoran la claridad y la facilidad del diseño.
- ✓ Simplifican el mantenimiento y las mejoras en funcionalidad.
- ✓ A menudo, se genera un bajo acoplamiento.
- ✓ Soporta una mayor capacidad de reutilización.

1.4.1.3 Creador

Permite identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases.

Asignarle a la clase B la responsabilidad de crear una instancia de la clase A. (3)

Beneficios:

Brinda soporte a un bajo acoplamiento, lo cual supone poca dependencia respecto al mantenimiento y mejores oportunidades de reutilización.

1.4.1.4 Experto

Este patrón permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada (disminución del acoplamiento). (3)

Ventajas:

- ✓ Se conserva el encapsulamiento.
- ✓ Soporta un bajo acoplamiento, lo que favorece al desarrollo de sistemas más robustos y de fácil mantenimiento.
- ✓ El comportamiento se distribuye entre las clases que cuentan con la información requerida. Brinda soporte a una alta cohesión.

1.4.1.5 Controlador

Posibilita la asignación de responsabilidades para controlar el flujo de eventos del sistema, a clases específicas.

Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario enviándolo a las distintas clases según el método llamado. (3)

Beneficios:

- ✓ Mayor potencial de los componentes reutilizables.
- ✓ Analiza sobre el estado del caso de uso.

1.4.2 Patrones Estructurales

Los patrones estructurales están relacionados con cómo las clases y los objetos se combinan para dar lugar a estructuras más complejas y describen las formas de componer los objetos para conseguir una nueva funcionalidad. (3)

1.4.2.1 Facade (Fachada)

Proporciona una interfaz unificada a un conjunto de interfaces de un sistema. Facade define una interfaz de alto nivel, posibilitando que el sistema sea más fácil de usar. (3)

Ventajas:

- ✓ Facilita el uso del subsistema.
- ✓ Se promueve un acoplamiento débil entre el subsistema y sus clientes, al ser eliminadas o reducidas las dependencias.
- ✓ No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten. De esta forma, se puede elegir entre facilidad de uso y generalidad.

1.4.3 Patrones de Comportamiento

Los patrones de comportamiento estudian las relaciones de llamadas entre los diferentes objetos, proporcionan estrategias comprobadas para modelar la manera en que los objetos colaboran entre sí en un sistema. (3)

1.4.3.1 Mediator (Mediador)

Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto. Evita las referencias explícitas entre los objetos y permite, por tanto, que su interacción se modifique de forma independiente. (3)

Ventajas:

- ✓ Al reducir el acoplamiento entre los objetos que interactúan es más fácil variar o reutilizar dichos objetos.

✓ Al hacer de la mediación un concepto independiente, encapsulado en un objeto a parte, se favorece la concentración de la atención en la interacción entre objetos, al margen del comportamiento individual de cada uno.

1.4.3.2 Strategy (Estrategia)

Dispone de varios métodos para resolver un problema y elige cuál utilizar en tiempo de ejecución. Este permite a los algoritmos variar con independencia de los clientes que los usan. (3)

Ventajas:

✓ Se eliminan las definiciones de comportamientos multicondicionales.

✓ Posibilita ofrecer diferentes implementaciones del mismo comportamiento, en función de restricciones como el espacio en memoria o el tiempo de respuesta.

Se concluye con que los patrones de diseño:

✓ Proponen una forma de reutilizar la experiencia de los desarrolladores, clasificando y describiendo formas de solucionar problemas que ocurren en el desarrollo de software.

✓ Están basados en la recopilación del conocimiento de los expertos en desarrollo de software.

1.5 Lenguajes de programación

Un lenguaje es diseñado para describir un conjunto de acciones que un equipo debe ejecutar. Está formado por un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Permiten principalmente el desarrollo de software, además de que es utilizado para controlar el comportamiento físico y lógico de una máquina.

1.5.1 Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado, comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La POO expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar. La POO maneja definiciones nuevas tales como: Clase, Objeto, Método, Evento, Mensaje, Atributo, Estado Interno, Componentes de un Objeto y Representación de un

Objeto, además de un conjunto de características propias como son: Abstracción, Encapsulamiento, Principio de Ocultación, y Polimorfismo. (4)

1.5.2 PHP

Hypertext Pre Processor es un lenguaje script del lado del servidor utilizado para la generación de páginas Web dinámicas, que es embebido dentro del código HTML. Es un lenguaje sencillo de sintaxis cómoda, es rápido y dispone de una gran cantidad de librerías que facilitan el desarrollo de aplicaciones. (5)

Ventajas:

✓ PHP corre en diversas plataformas utilizando el mismo código fuente, pudiendo ser compilado y ejecutado en 25 plataformas, incluyendo diferentes versiones de Unix, Windows (95, 98, NT, ME, 2000, XP, Vista, Seven) y Mac.

✓ La sintaxis de PHP es similar a la del C, por esto cualquiera con experiencia en lenguajes del estilo C podrá entender rápidamente PHP. Entre los lenguajes del tipo C se incluyen al Java y JavaScript.

✓ PHP es completamente expandible. Está compuesto de un sistema principal, un conjunto de módulos y una variedad de extensiones de código.

✓ Muchas interfaces distintas para cada tipo de servidor. Este lenguaje actualmente se puede ejecutar bajo Apache, IIS, AOLServer, Roxen yTHTTPD. Otra alternativa es configurarlo como módulo CGI.

✓ Puede interactuar con numerosos motores de bases de datos tales como MySQL, SQL, Oracle, Informix, PostgreSQL.

✓ PHP generalmente es utilizado como módulo de Apache, lo que lo hace extremadamente veloz. Está completamente escrito en C, así que se ejecuta rápidamente utilizando poca memoria.

✓ PHP es Open Source, lo cual significa que el usuario no depende de una compañía específica para arreglar cosas que no funcionan, además no está forzado a pagar actualizaciones anuales para tener una versión del producto.

1.5.3 XHTML

El XHTML acrónimo en inglés de e**X**tensible **H**yper**t**ext **M**arkup **L**anguage (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir al HTML como estándar para las páginas Web. (6)

Está encaminado al uso de un etiquetado correcto, por lo que exige una serie de requisitos básicos a cumplir en cuanto al código. Algunos de estos requisitos son:

- ✓ Elementos correctamente anidados.
- ✓ Etiquetas en minúsculas.
- ✓ Elementos cerrados correctamente.
- ✓ Atributos de valores entrecomillados.

1.5.4 CSS

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. Entre las características de este lenguaje se encuentra la separación de los contenidos de su presentación, siendo esto imprescindible para crear páginas Web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML o XHTML bien definidos y con significado completo (también llamados "documentos semánticos"). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes. (7)

Otro aspecto importante es que permite darle color, tamaño y tipo de letra al texto, además de posibilitar una separación horizontal y vertical entre elementos.

1.5.5 JavaScript

Es un lenguaje de tipo script compacto, basado en objetos y guiado por eventos, diseñado específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet. Además es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. Los programas JavaScript van incrustados en los documentos XHMTL o en un js, y se encargan de realizar acciones en el cliente, como pueden ser pedir datos, confirmaciones, mostrar mensajes, crear animaciones, comprobar campos. (8)

Ventajas:

- ✓ Los programas escritos en este lenguaje no requieren de mucha memoria ni tiempo adicional de transmisión, por ser pequeños y compactos.
- ✓ No requiere un tiempo de compilación; ya que los scripts se pueden desarrollar en un período de tiempo relativamente corto.
- ✓ Es independiente de la plataforma hardware o sistema operativo, y funciona correctamente siempre y cuando exista un navegador que lo soporte.
- ✓ Asegura la permanencia de una operación realizada, y aunque falle el sistema esta no podrá deshacerse.

1.5.5.1 JSON

Acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso del lenguaje de marcas extensible (XML). Es muy sencillo de usar, especialmente como alternativa a XML. Una de sus ventajas sobre XML como formato de intercambio de datos es que prevalece un formato mucho más sencillo a la hora de escribir un analizador semántico del mismo.

En JavaScript, JSON puede ser analizado trivialmente usando el procedimiento eval [1], lo cual ha sido fundamental para su aceptación por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador Web. (8)

1.5.6 XML

Es el estándar de Extensible Markup Language (Lenguaje de Etiquetado Extensible), conformado por un conjunto de reglas para definir etiquetas semánticas orientadas a organizar un documento en diferentes partes. Permite al usuario definir sus propios lenguajes de anotación adaptados a sus necesidades y contiene tres características muy importantes que son: extensibilidad, estructura y validación. (9)

Ventajas de XML:

- ✓ Las aplicaciones se pueden generar rápidamente y su mantenimiento es sencillo.
- ✓ Separa los datos de la presentación y del proceso, lo que permite mostrar y procesar los datos al gusto deseado con sólo aplicar distintas hojas de estilo y aplicaciones.
- ✓ La información es más accesible y reutilizable, por la flexibilidad de las etiquetas de XML que permiten su utilización sin tener que amoldarse a reglas específicas de un fabricante.
- ✓ Ofrece un formato para la descripción de datos estructurados, facilitando declaraciones de contenido más precisas y resultados de búsquedas más significativos en varias plataformas.

[1] Es una función para analizar si JSON permite una buena transferencia de datos.

1.6 Tecnología AJAX

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML) es una tecnología que facilita la creación de aplicaciones interactivas en la Web que se ejecutan en el navegador de los usuarios y mantienen comunicación asíncrona con el servidor; posibilitando que se puedan efectuar cambios sobre una página sin necesidad de recargarla, aumentando de esta manera la interactividad, velocidad y usabilidad de la misma. (10)

AJAX está conformado por:

- ✓ XHTML y CSS: para crear una presentación basada en estándares.
- ✓ DOM: para la interacción y manipulación dinámica de la presentación.
- ✓ XML, JSON: para el intercambio y la manipulación de información.
- ✓ XMLHttpRequest: para el intercambio asíncrono de información.
- ✓ JavaScript: para unir todas las demás tecnologías.

Además:

- ✓ Provee un mecanismo para mezclar y hacer coincidir XML con XHTML.
- ✓ Las aplicaciones son más rápidas e interactivas, al estilo aplicaciones de escritorio.
- ✓ Reduce de manera significativa tener que cargar información continuamente del servidor, actualizando solamente porciones de la página.
- ✓ Cuando se utiliza AJAX adecuadamente en el desarrollo de una aplicación, se reduce de manera significativa los tiempos de carga inicial.

1.7 Librería Extjs

Extjs es una librería de componentes que facilita las herramientas necesarias para la creación de aplicaciones Web. Es una interfaz de programación de aplicaciones (API) escrito en JavaScript con la finalidad de asistir el desarrollo de aplicaciones enriquecidas para internet.

En esta librería resalta un fuerte paradigma basado en componentes soportado por extensiones para la POO en JavaScript que facilitan la implementación de extensiones y aplicaciones de gran complejidad, el poseer un conjunto de widgets amplio, configurable y de muy alta calidad así como una implementación transparente y sencilla para el trabajo con AJAX. Es importante destacar que uno de los mayores éxitos API está en la amplia, detallada y bien elaborada documentación, en la cantidad de ejemplos y en la poderosa comunidad detrás de este desarrollo. (11)

Actualmente Extjs es considerado un FrameWork independiente, ya que a principios del 2007 se creó una compañía para comercializar y dar soporte al mismo. Dicha

compañía proporciona los servicios de consultoría necesarios para ayudar a los clientes en el aprovechamiento máximo de las ventajas de Extjs.

Extjs brinda soporte para diversos navegadores como: Internet Explorer, Firefox, Safari y Opera. Entre sus principales ventajas se encuentran:

- ✓ Comunica datos de forma asincrónica con el servidor.
- ✓ Construye interfaces gráficas complejas y dinámicas.

1.8 Herramientas

La selección adecuada de herramientas a utilizar en el desarrollo de un software tiene estrecha relación con el tiempo de desarrollo y la calidad final del producto. Todas las herramientas utilizadas en la realización del módulo fueron definidas por el centro. Es por ello que en este epígrafe no se definen los instrumentos utilizados sino que se realiza una identificación de cada uno de ellos y las ventajas que proporciona en el trabajo de diseño e implementación del módulo.

1.8.1 Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. (13)

Está diseñado para desarrollar software con POO. Dentro de sus características fundamentales se encuentran:

- ✓ Multiplataforma: Soportada en plataformas Java para Sistemas Operativos Windows, Linux y Mac OS X.
- ✓ Interoperabilidad: Intercambia diagramas UML y modelos con otras herramientas. Soporta exportar e importar a XMI19, XML y archivos Excel. Importa archivos de proyectos de Rational Rose. Integración con Microsoft Office Visio.
- ✓ Modelamiento de los Requisitos: Captura de requisitos con diagrama de requisitos, modelamiento de casos de uso y análisis textual.
- ✓ Colaboración de Equipo: Realiza el modelado simultáneamente con el Visual Paradigm TeamWork Server y Subversion.
- ✓ Generación de Documentación: Comparte y genera los diagramas y diseños en formatos como PDF20, HTML y Microsoft Word.

✓ Editor de Detalles de Casos de Uso: Entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.

✓ Ingeniería de Código: Permite generación de código e ingeniería inversa en lenguajes como Java, C, PHP, XML, Python, C#, VB .NET, Flash, ActionScript, Delphi y Perl.

✓ Modelado de Procesos de Negocio: Visualiza, comprende y mejora los procesos de negocio con la herramienta para estos procesos.

✓ Integración con Entornos de Desarrollo: Apoyo al ciclo de vida completo de desarrollo del software: análisis, diseño e implementación, en IDE como Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, JBuilder y otros.

✓ Modelamiento de Bases de Datos: Generación de bases de datos y conversión de diagramas entidad -relación a tablas de base de datos, además de mapeos de objetos y relaciones.

Se utilizó esta herramienta porque soporta el ciclo de vida completo, además de las facilidades que brinda para el modelado de los diferentes diagramas UML.

1.8.2 PGadmin III

PGAdmin III es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL, siendo la más completa con licencia Open Sourse. Está escrita en C++ usando la librería gráfica multiplataforma Widgets, lo que permite que se pueda usar en Linux, FreeBSD, Solaris, Mac OS X y Windows. (14)

1.8.3 ZendEstudio for Eclipse

ZendStudio 6.0 es un IDE de desarrollo para aplicaciones de la Web 2.0, gratuito y de código abierto. Está basado en el conocido entorno de desarrollo Eclipse, también de código abierto; pero mientras que Eclipse está focalizado en el desarrollo para Java, ZendStudio es una distribución focalizada en el desarrollo Web, con soporte a HTML, CSS y JavaScript.

Soporta varias librerías como: ExtJS, Yahoo UI y JQuery para la presentación así como la inclusión de FrameWork para el desarrollo de la Web pudiendo combinarlas fácilmente en una aplicación. ZendStudio está disponible como una aplicación independiente, se puede encontrar para dos plataformas fundamentales: Windows y GNU/Linux.

Esta herramienta fue seleccionada para el desarrollo porque permitió la programación del módulo integrándose con marco de trabajo del CEDRUX y por su facilidad de completamiento del código principalmente con el lenguaje PHP.

1.8.4 Navegador

Mozilla Firefox 3.* es el nuevo e innovador navegador open source. Firefox ha sido creado por el proyecto Mozilla, un esfuerzo open source sin ánimo de lucro que incluye a miles de voluntarios alrededor del mundo. La misión del proyecto Mozilla es preservar la elección y la innovación en Internet. El apoyo organizativo del proyecto Mozilla es proporcionado por Mozilla Foundation (en los Estados Unidos de América), Mozilla Europe y Mozilla Japón.

Ventajas:

- ✓ Navegación por pestañas.
- ✓ Búsqueda progresiva.
- ✓ Corrector ortográfico
- ✓ Marcadores dinámicos
- ✓ Administrador de descargas y un sistema de búsqueda integrado que utiliza el motor de búsqueda que desee el usuario.

Algunas de las características de Mozilla Firefox 3.* son:

- ✓ Multiplataforma.
- ✓ Cuenta con una protección antiphishing, antimalware e integración con el antivirus.
- ✓ Múltiples Extensiones.
- ✓ Incluye un buscador integrado en la interfaz que hace búsquedas en Google.

1.8.5 SVN (Subversion)

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es un software libre bajo la licencia Apache y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente; en cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. (15)

Existen dos razones fundamentales para el uso de esta herramienta:

- ✓ Gestiona las modificaciones durante el desarrollo.

✓ Permite que varias personas trabajen sobre los mismos ficheros.

1.8.6 Sistema Operativo

Ubuntu es una distribución Linux que ofrece un sistema operativo predominantemente enfocado a ordenadores de escritorio, aunque también proporciona soporte para servidores. Basada en Debian GNU/Linux, Ubuntu concentra su objetivo en la facilidad y la libertad en la restricción de uso, los lanzamientos regulares (cada 6 meses) y la facilidad en la instalación.

1.9 Frameworks

Los frameworks orientados al objeto son la piedra angular de la moderna ingeniería del software. El desarrollo de frameworks está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente (source code). Los frameworks son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. (16)Un framework no tiene funcionalidades de una aplicación específica, sino que las aplicaciones se construyen sobre ellos.

1.9.1 Zend Framework

Es un framework para desarrollo de aplicaciones Web y servicios Web con PHP. Brinda soluciones para construir sitios Web modernos, robustos y seguros. Además, es Open Source y trabaja con PHP 5.

Zend Framework utiliza el estilo Modelo Vista Controlador (MVC) como base de su funcionamiento y se caracteriza por ser integrable a las aplicaciones debido a su composición y a que contiene diferentes clases de gran utilidad, como por ejemplo en la búsqueda dinámica de ficheros a incluir o utilizar. Cuenta con un importante mecanismo de manejo de controladores y vistas. (17)

Dentro de sus principales características están:

✓ Proporciona los componentes que forma la infraestructura del patrón MVC.

✓ Proporciona mecanismos de filtrado y validación de entradas de datos.

✓ Permite convertir estructuras de datos PHP a JSON y viceversa, para su utilización en aplicaciones AJAX.

✓ Incluye objetos de las diferentes bases de datos, por lo que es extremadamente simple para consultar la base de datos.

✓ Completa documentación y pruebas de alta calidad.

- ✓ Robustas clases para autenticación y filtrado de entrada.
- ✓ Proporciona un sistema de caché dividido en frontend y backend, de forma que se puedan almacenar en caché diferentes datos como resultados de funciones y páginas completas.

1.9.2 Doctrine Framework

Doctrine es un mapeador de objetos relacionales (ORM) para PHP 5.2.3 que se apoya en una potente capa de abstracción de bases de datos. Tiene la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir, convertir clases (convenientemente creadas siguiendo las pautas del ORM) a tablas de una base de datos. (18)

1.9.2.1 ORM

Object Relational Mapper es una técnica de programación que nos permite convertir datos entre el sistema de tipos, utilizado en un lenguaje de POO y en una base de datos relacional, es decir, las tablas de la base de datos pasan a ser clases y los registros objetos que se pueden manejar con facilidad.

1.9.3 Cedrux Framework

Este marco de trabajo es utilizado en la UCI. El mismo está montado sobre Zend y Doctrine framawork. Consta de varios módulos que son desarrollados con la librería Extjs. En él se aplica la arquitectura modelo vista controlador lo que lo hace más rápido y funcional; permite además una gran integración con otros módulos que se desarrollan en la universidad.

1.10 Arquitectura

Una arquitectura de software consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para su sistema de información.

1.10.1 Modelo Vista Controlador

El patrón de diseño Modelo Vista Controlador describe una forma utilizada en la Web, de organizar el código de una aplicación separándolos en tres componentes distintos. (19)

Modelo: Componente encargado del acceso a datos.
Vista: Define la interfaz de usuario, HTML, CSS, enviados en el navegador.
Controlador: Responde a eventos y modifica la vista y el modelo.
La principal ventaja de este patrón es la facilidad para realizar cambios en la aplicación, ya que cuando se realiza un cambio de bases de datos, programación o interfaz de usuario, estos componentes son tratados de forma independiente.

1.10.2 Arquitectura Cliente Servidor

Esta arquitectura se divide en dos partes claramente diferenciadas, la primera es la parte del servidor y la segunda está relacionada con los clientes.
Normalmente, el servidor es una máquina potente que actúa de depósito de datos y funciona como un sistema gestor de base de datos (SGBD). Por otro lado, los clientes suelen ser estaciones de trabajo que solicitan varios servicios al servidor.
Ambas partes deben estar conectadas entre sí mediante una red. (19)
La misma necesita tres tipos de software para su correcto funcionamiento:
✓ Software de gestión de datos: Se encarga de la manipulación y gestión de los datos almacenados y requeridos por las diferentes aplicaciones. Se aloja en el servidor.
✓ Software de desarrollo: Se aloja en los clientes y en aquellos que se dediquen al desarrollo de aplicaciones.
✓ Software de interacción con los usuarios: También reside en los clientes y es la aplicación gráfica de usuario para la manipulación de datos, siempre a nivel de usuario (consultas principalmente).

1.11 Conclusiones

En este capítulo se hizo una descripción de las principales herramientas, tecnologías y metodología propuesta en el proyecto, sentando las bases en el desarrollo del producto. Las tecnologías y los frameworks a utilizar contienen tendencias recientes de la programación, por los que las versiones son cada vez más usables y fáciles.
En la confección de dicha aplicación se utilizó como lenguaje de programación del lado del servidor PHP 5.2, como servidor Web Apache, como gestor de base de datos PostgreSQL, la programación por el lado del cliente XHTML, JavaScript, CCS. Además, se utilizó las herramientas Zend Studio y Visual Paradigm para el desarrollo y creación de los diagramas UML respectivamente.

Capítulo 2: Diseño de la Solución

2.1 Introducción

En el presente capítulo se describirán los elementos fundamentales a tener en cuenta en el desarrollo del producto. Se realizarán los diferentes diagramas de clases, de interacción y los prototipos de interfaz para posibles implementaciones.

2.2 Propósito del Diseño

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción, lo cual contribuye al desarrollo de una arquitectura estable y sólida. Este flujo de trabajo es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales.

La realización del diseño permite ser una guía para que los desarrolladores puedan leer y entender el producto que se desea construir. En él se definen las clases y asociaciones que se utilizarán en la implementación, así como las interfaces y los algoritmos de los métodos utilizados para implementar las operaciones.

Los objetivos del diseño son:

- ✓ Desarrollar un diseño para el sistema.
- ✓ Adaptar el diseño a las particularidades del sistema para que sea consistente con el entorno de implementación.

2.3 Diagrama de Casos de uso del Sistema

Un caso de uso es una secuencia de actividades que realiza un sistema y que da como resultado un valor para el actor. Estos han alcanzado un uso universal debido a dos razones básicas, la primera de ellas es que proporcionan un medio intuitivo y sistemático de capturar los requisitos anteriormente mencionados, centrándose en lo que quiere obtener el cliente, y la segunda es que dirigen todo el proceso apreciando que el análisis, diseño y prueba se realizan partiendo de los casos de uso. Es de vital importancia realizar una buena selección de los casos de uso debido a que el proceso de desarrollo está guiado por ellos, lo que se traduce en que, una serie de flujos de trabajo se inicia a partir de los mismos. A continuación se expone el diagrama de casos de uso del sistema, donde hay una serie de artefactos que son fundamentales para el diseño y la implementación del módulo de Gestión de indicadores.

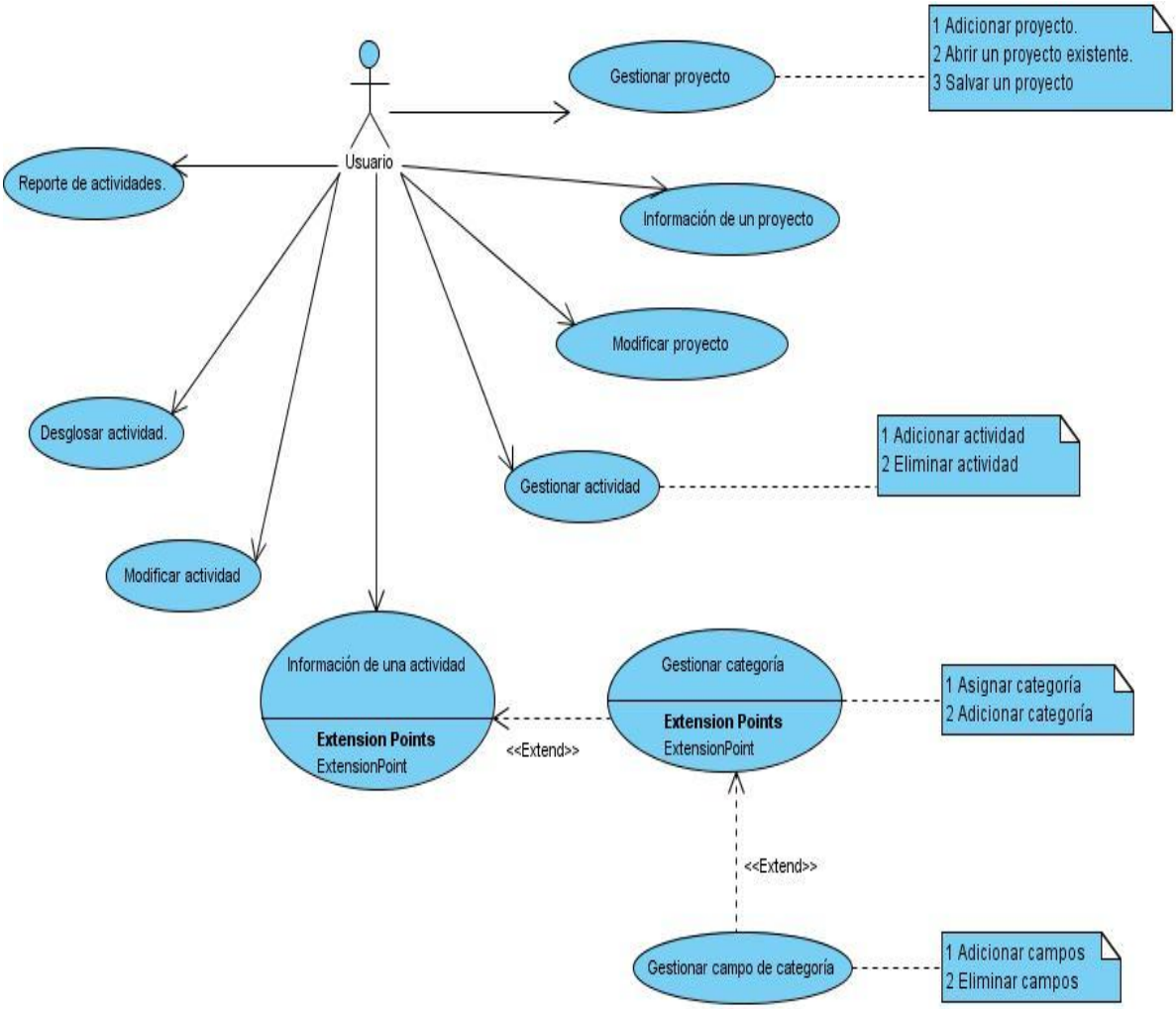


Figura 1: Diagrama de casos de uso del sistema.

2.4 Requisitos funcionales

Para el diseño e implementación del módulo de Gestión de indicadores se tomaron en cuenta una serie de requisitos funcionales que a continuación se hace referencia brevemente.

- ✓ R1 Adicionar proyecto.
- ✓ R2 Abrir un proyecto existente.
- ✓ R3 Salvar un proyecto.
- ✓ R4 Mostrar información de un proyecto.
- ✓ R5 Modificar información del proyecto.
- ✓ R6 Adicionar actividad.
- ✓ R7 Eliminar actividad.
- ✓ R8 Modificar actividad.
- ✓ R9 Mostrar información de una actividad.

- ✓ R10 Adicionar nueva categoría.
- ✓ R11 Asignar categoría a una actividad.
- ✓ R12 Adicionar campo personalizado a una categoría.
- ✓ R13 Eliminar campos personalizados a una categoría.
- ✓ R14 Desglosar actividad.
- ✓ R15 Modificar el valor de la categoría.
- ✓ R16 Modificar categoría.
- ✓ R17 Mostrar información de la categoría.
- ✓ R18 Reporte de actividades.
- ✓ R19 Mostrar la fecha más antigua.
- ✓ R20 Mostrar la máxima fecha.

2.5 Requisitos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades son las características que hacen al producto atractivo, usable, rápido y confiable.

2.5.1 Usabilidad

Permite mostrar la información de forma lógica y puede ser usado por cualquier persona con conocimientos básicos sobre el uso de la computadora y un ambiente Web en sentido general.

2.5.2 Fiabilidad

El sistema se debe recuperar en un tiempo prudencial de acuerdo con la anomalía ocurrida.

2.5.3 Rendimiento

La aplicación debe soportar la conexión concurrente de 1.000 peticiones en una ventana de 1s, destacando que los tiempos de respuesta no deben exceder de los 400ms ± 100ms para el 60% de las peticiones de consulta y para las peticiones de escritura no deben exceder de los 900ms ± 400ms.

2.5.4 Soporte

La aplicación contará antes de su puesta en marcha con un período de pruebas, se le dará mantenimiento, configuración y se brindará el servicio de instalación.

2.5.5 Restricciones de diseño

El software contará con las siguientes restricciones:

✓ El lenguaje de programación a utilizar es el PHP.

✓ Se utilizará framework de trabajos como el Zend Framework, ERP y Doctrine para el desarrollo.

✓ No se debe usar funciones propias de algún sistema operativo.

✓ Se utilizará Visual Paradigm for UML como herramienta CASE para el modelado y obtención de los diferentes artefactos que requiere el software.

2.5.6 Interfaz

La aplicación contendrá distintas interfaces que servirán para el intercambio de información con la misma.

✓ Las interfaces de usuario implementadas serán Web.

✓ La interfaz de este sistema debe contar con un diseño sencillo y al mismo tiempo permitir la interpretación correcta de la información cumpliendo con los siguientes requisitos:

- Las ventanas del sistema contendrán claro y bien estructurado los datos, y al mismo tiempo permitirán la interpretación correcta e inequívoca de la información.
- El diseño de la interfaz de usuario estará orientado a la ejecución de acciones de una manera rápida, minimizando los pasos a dar en cada proceso.
- Todos los textos y mensajes en pantalla aparecerán en idioma español.

✓ El Sistema se comunicará con los gestores de bases de datos mediante protocolo TCP/IP.

✓ Los clientes accederán al sistema vía HTTP.

2.5.7 Disponibilidad

El sistema deberá estar disponible 24x7.

2.5.8 Hardware

El producto informático requiere de un servidor de 2 GB de RAM como mínimo y 4 GB de espacio libre en disco duro. Todas las computadoras implicadas, tanto para la administración como las de los usuarios, deben estar conectadas a una red y tener al menos 1 GB de RAM. Además, contará con una conexión de red de 54 Mbps entre el cliente y el servidor.

2.5.9 Software

- ✓ El sistema debe ser desarrollado en lenguaje de script PHP 5 o superior.
- ✓ Para la instalación del servidor:
  - Sistema Operativo: Linux Ubuntu 9.10 (recomendado o superior) o Microsoft Windows.
  - Servidor Web Apache 2.
  - Servidor de Base de Datos Postgresql 8.4
- ✓ Para interpretación por el cliente:
  - Sistema Operativo: Linux Ubuntu 9.10 (recomendado o superior) o Microsoft Windows.
  - Navegadores: Internet Explorer v6, Internet Explorer v7 (recomendado) o superior, Mozilla Firefox 2.* (recomendado) o superior (recomendado).

2.6 Diagramas de clases del diseño

En un diagrama de clases de diseño se muestran los atributos y métodos de cada clase y se representa de forma sencilla la colaboración y las responsabilidades de las distintas clases que forman el sistema. Se utilizan para modelar la vista de diseño estática de un sistema, esto incluye el vocabulario del sistema, las colaboraciones o esquemas. (20)

Una clase de diseño es una construcción similar en la implementación del sistema. El lenguaje utilizado para especificar estas clases es el lenguaje de programación que se utilizará para la implementación. Las operaciones, atributos, tipos, visibilidad (public, protected, private), se pueden especificar con la sintaxis del lenguaje elegido. Los métodos del diseño tienen correspondencia directa con los métodos de implementación.

Una clase de diseño puede proporcionar interfaces si tiene sentido hacerlo en el lenguaje de programación. El diseño obtenido cumple con los patrones de Bajo acoplamiento y Alta cohesión permitiendo la colaboración entre los elementos de las clases, sin verse afectados la reutilización de los mismos y el entendimiento de estos cuando se encuentran aislados. A cada clase le fueron asignadas las tareas que podían realizar según la información que poseían, además de crear las instancias de otras clases en correspondencia con la responsabilidad dada; poniéndose de manifiesto los patrones Experto y Creador. Además, el patrón controlador se usa para asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas.

Los siguientes epígrafes muestran los diferentes diagramas de clases, diseñados para la implementación del sistema, para consultar el resto ver anexo 1.

2.6.1 Diagrama de clases del Caso de Uso Gestionar Actividad

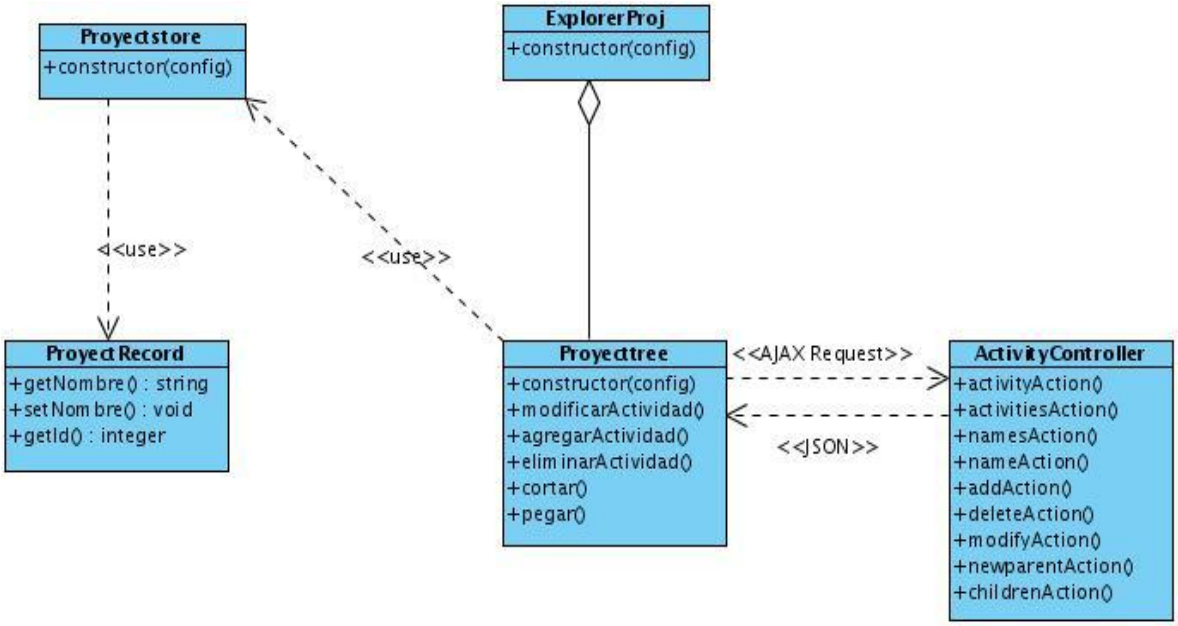


Figura 2: Diagrama de clases CU_Gestionar actividad.

2.6.2 Diagrama de clases del Caso de Uso Gestionar Categoría

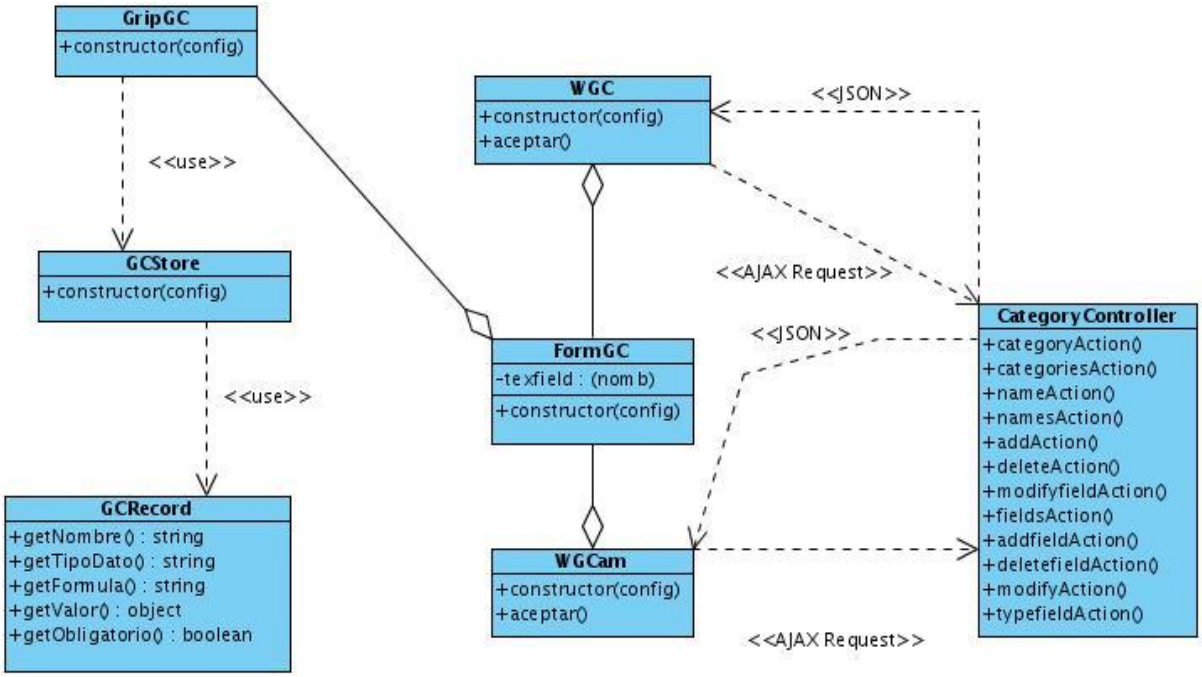


Figura 3: Diagrama de clases del CU_Gestionar Categoría.

2.6.3 Diagrama de clases del Caso de Uso Gestionar Proyecto

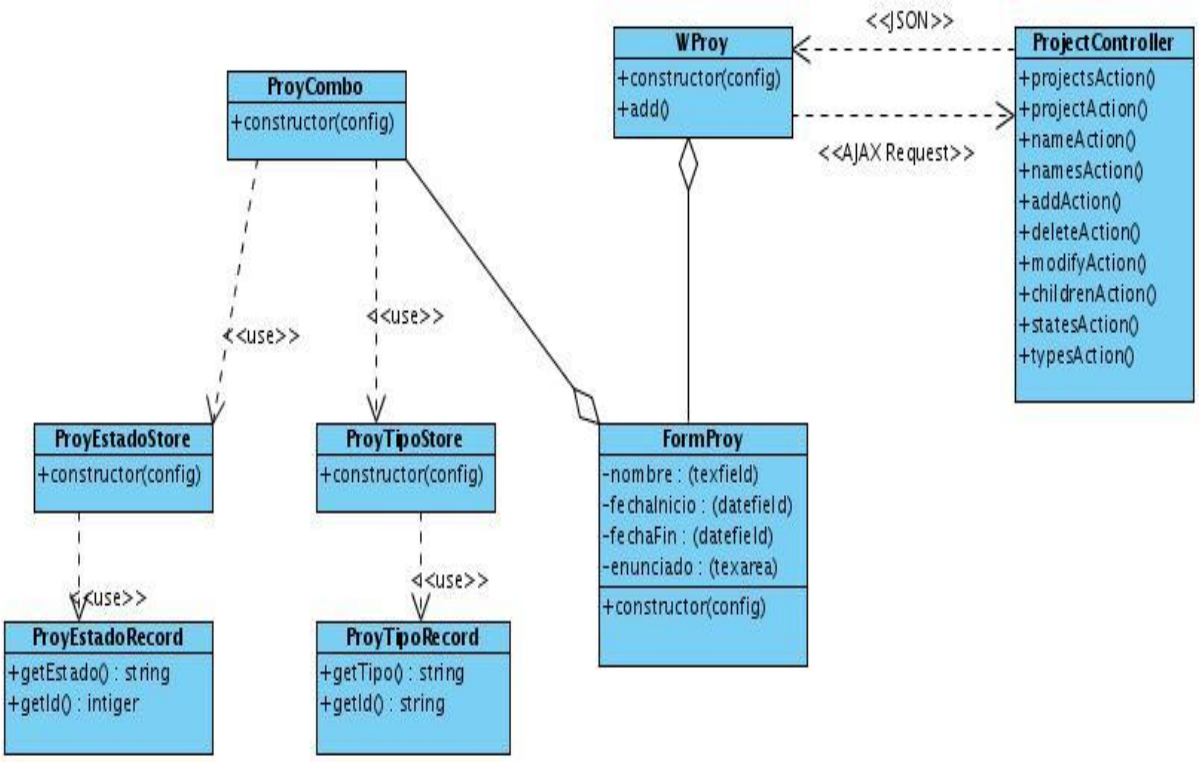


Figura 4 : Diagrama de clases del CU_Gestionar Proyecto.

2.6.4 Diagrama de clases general

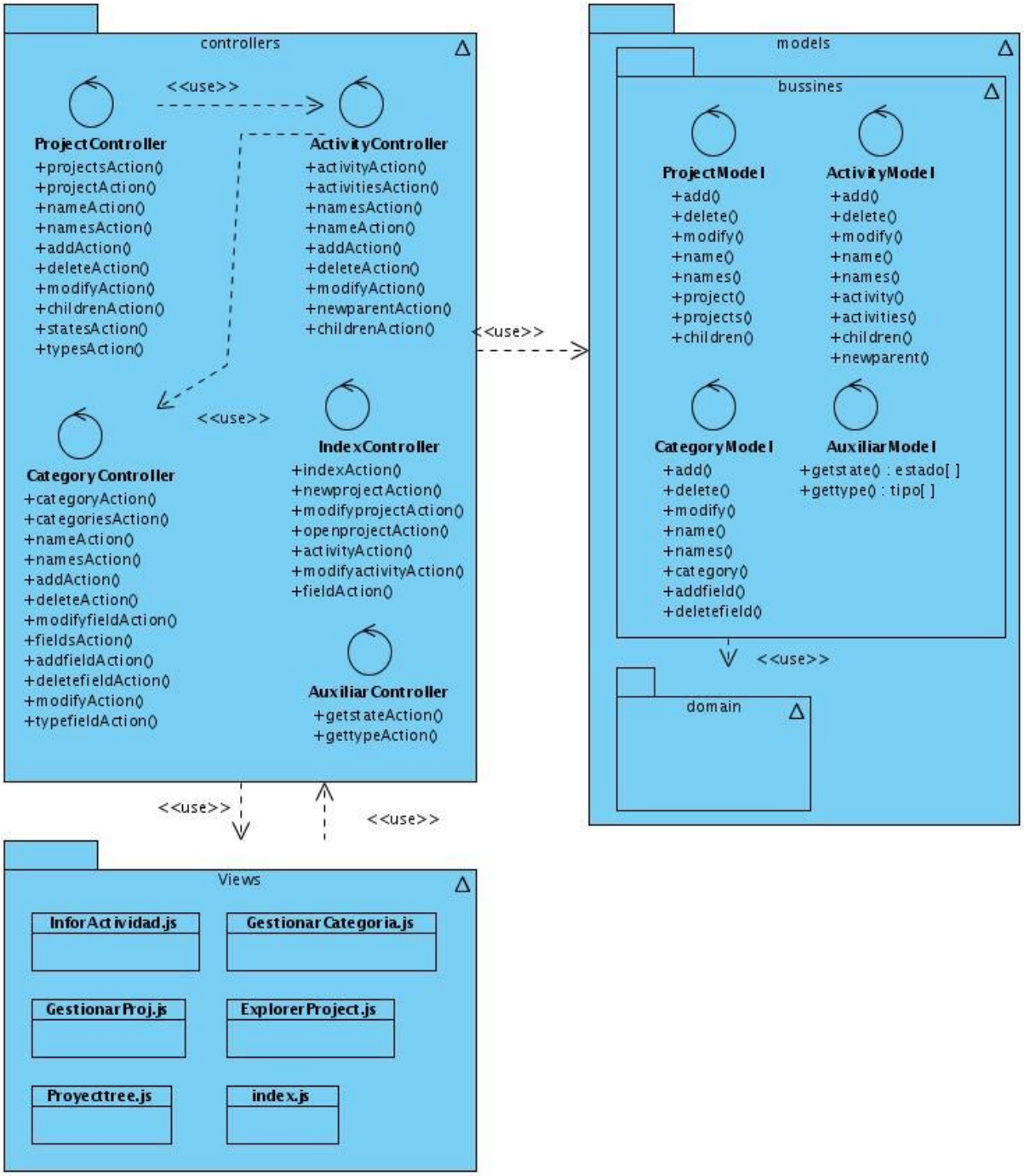


Figura 5 : Diagrama de clases general

2.7 Diagramas de interacción del diseño

Los diagramas UML llamados diagramas de interacción (secuencia y colaboración) se utilizan para modelar los aspectos dinámicos de un sistema lo que conlleva modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento. (21)

Un diagrama de secuencia muestra una interacción que está organizada como una secuencia temporal. En particular, muestra los objetos que participan en la interacción mediante sus líneas de vida y mediante los mensajes que intercambian, organizados en forma de una secuencia temporal. Un diagrama de secuencia no muestra los enlaces existentes entre objetos y tienen distintos formatos adecuados para propósitos diferentes. A continuación se muestran algunos de los diagramas de secuencias, para consultar el resto ver anexo 2.

2.7.1 Diagrama de secuencia del Caso de Uso Gestionar Actividad

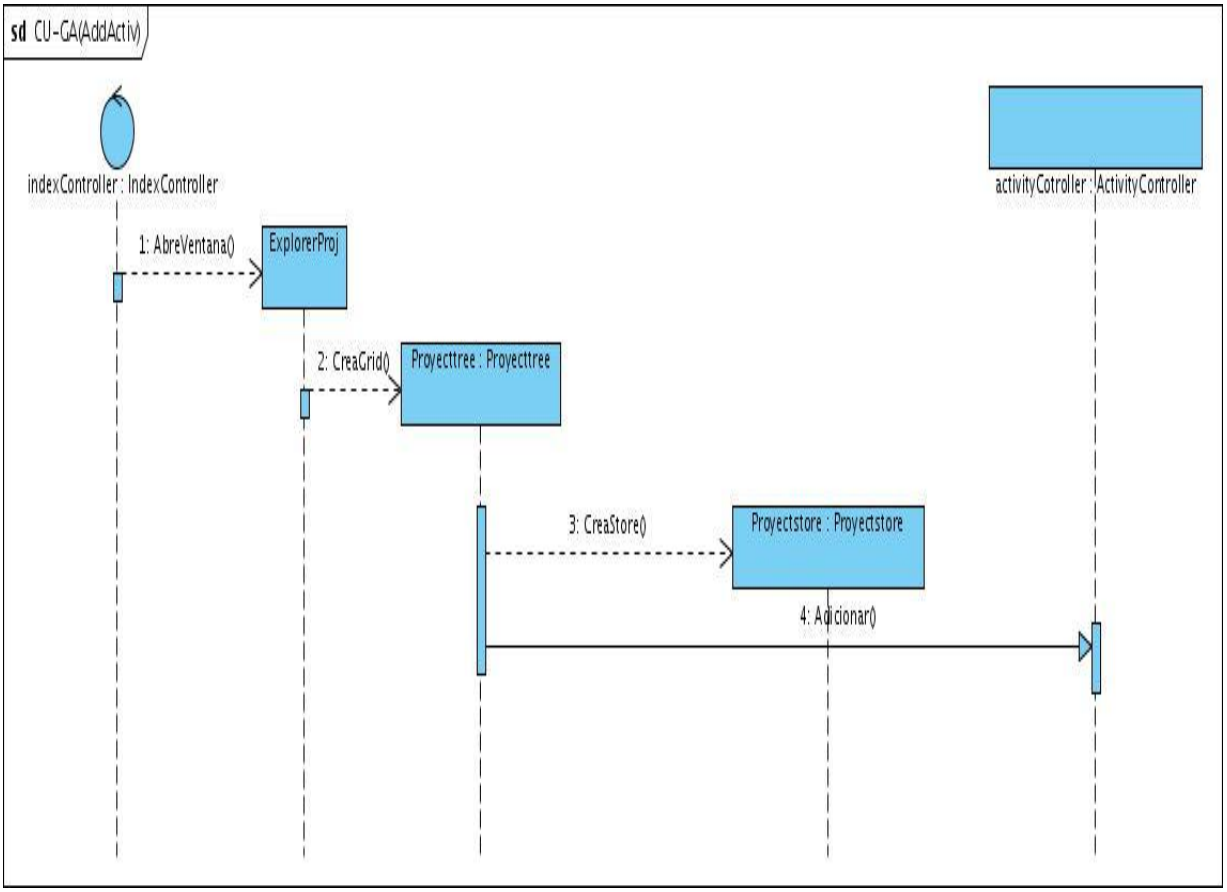


Figura 6 : Diagrama de secuencia del CU_Gestionar Actividad.

2.7.2 Diagrama de secuencia del Caso de Uso Gestionar Categoría

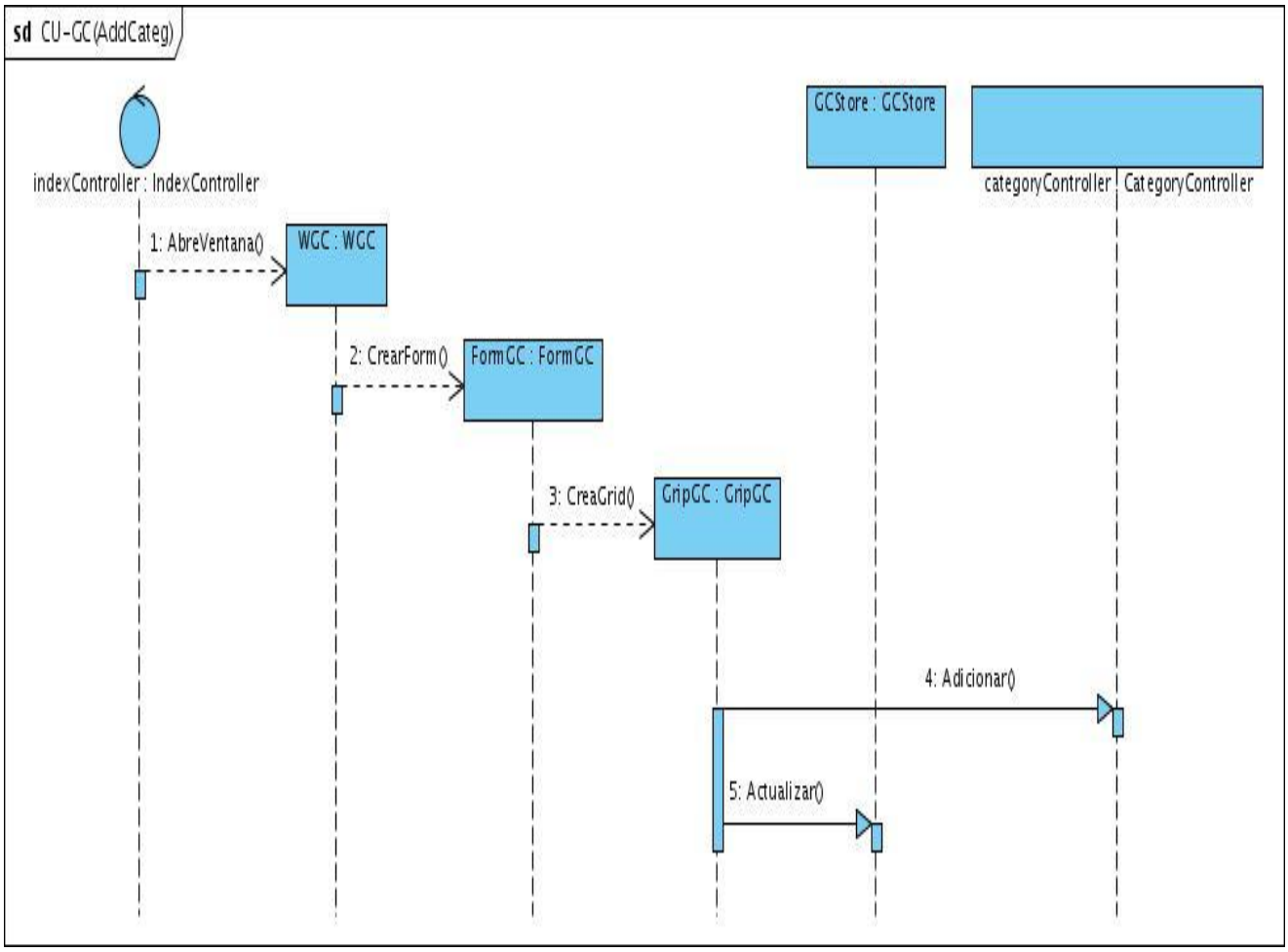


Figura 7 : Diagrama de secuencia CU_Gestionar Categoría.

2.7.3 Diagrama de secuencia del Caso de Uso Gestionar Proyecto

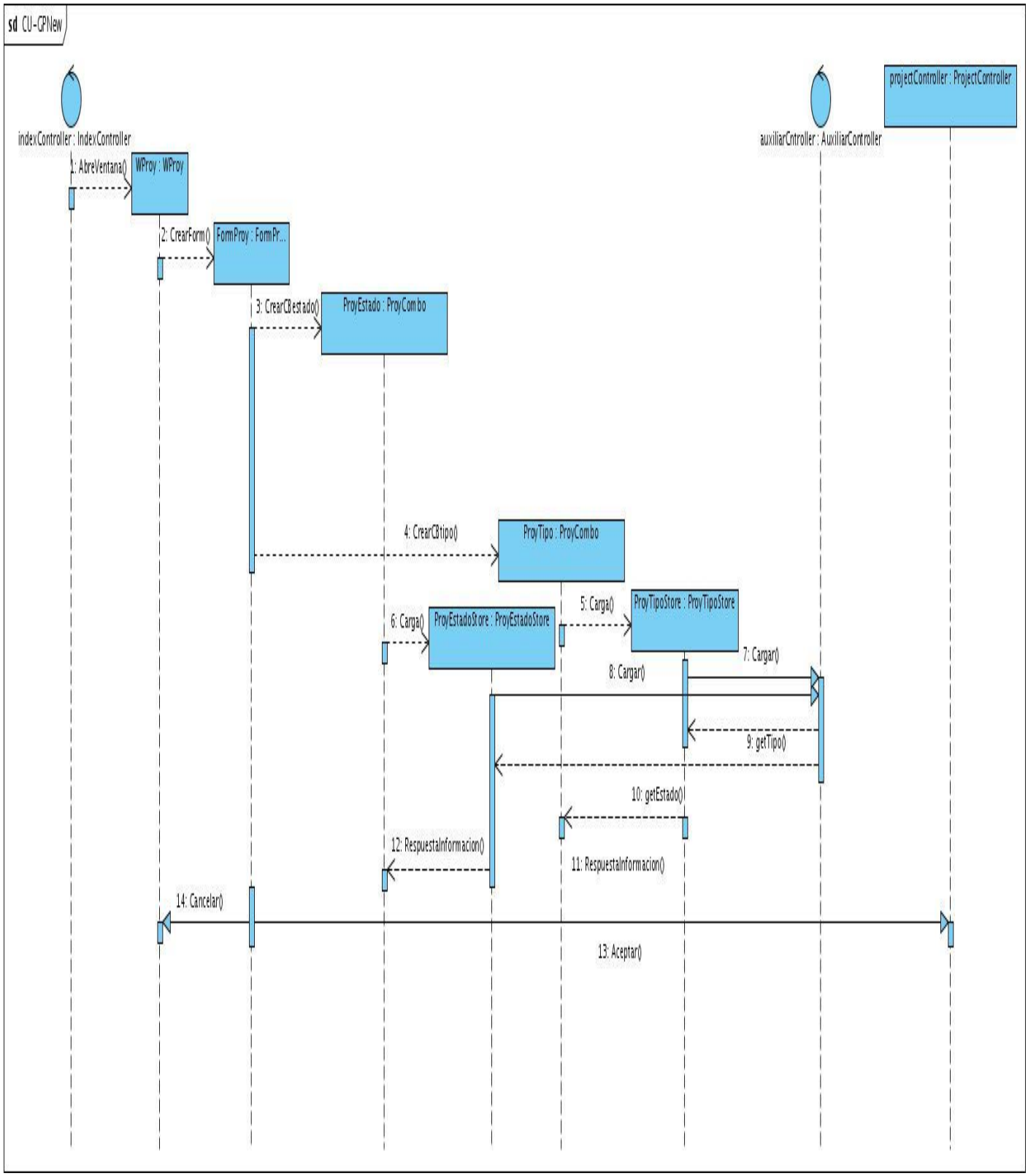


Figura 8 : Diagrama de secuencia CU_Gestionar Proyecto.

2.8 Prototipos de interfaz de usuario.

En este epígrafe se relacionan los requisitos funcionales establecidos por cada componente en cada escenario. A continuación se muestran los diferentes prototipos diseñados en el Visual Paradigm.

2.8.1 Prototipo de interfaz para Adicionar proyecto

En el Módulo gestión de indicadores una de las actividades más importantes que se realiza es la de adicionar un nuevo proyecto, teniendo en cuenta que este es el encargado de garantizar que se creen los diferentes proyectos con que contará el MINCI. En este prototipo se recoge además el requisito funcional R3 Salvar un proyecto.

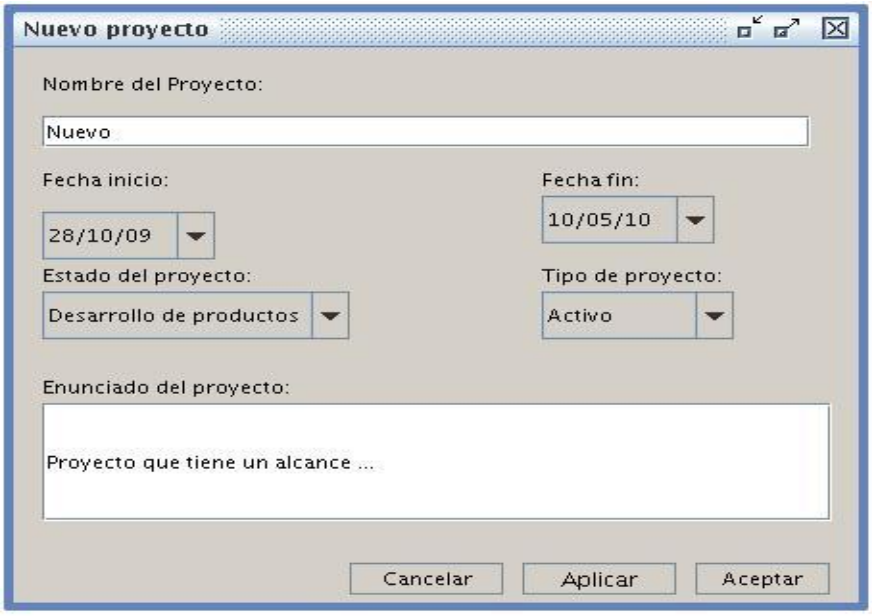


Figura 9: Prototipo de interfaz para requisitos R1, R3.

2.8.2 Prototipo de interfaz para la información del proyecto

El usuario puede modificar el proyecto, por lo que el sistema brinda la posibilidad de modificarlo, y además el prototipo recoge el requisito funcional R4 Mostrar información de un proyecto.

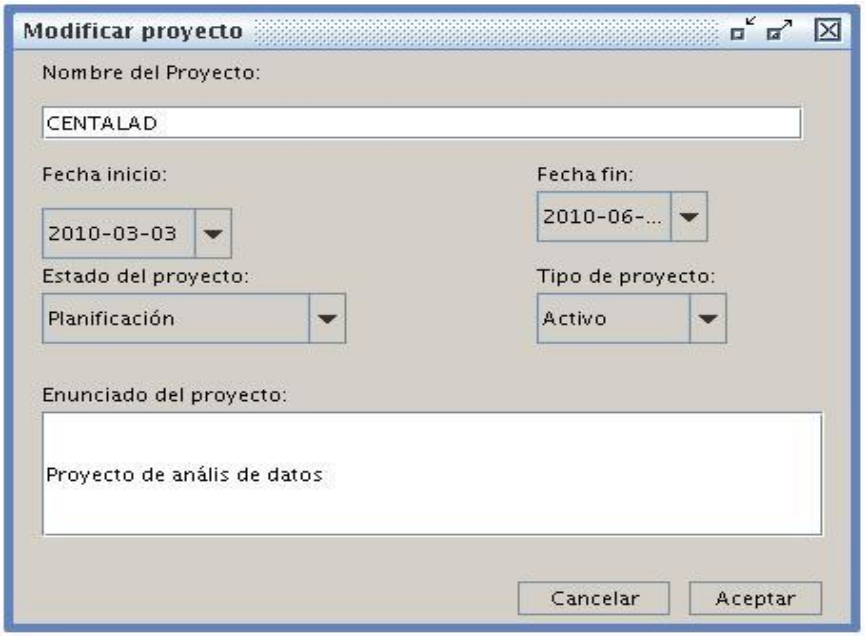


Figura 10: Prototipo de interfaz para requisitos R4, R5.

2.8.3 Prototipo de interfaz para Adicionar la actividad

El sistema permite agregar una actividad al proyecto en edición y también actividades dentro de otras. Este prototipo recoge los requisitos funcionales R7 Eliminar actividad, R14 Desglosar actividad y R8 Modificar actividad, este último se realiza interactuando sobre la actividad o sus valores y se puede modificar, salvándose automáticamente en la base de datos.

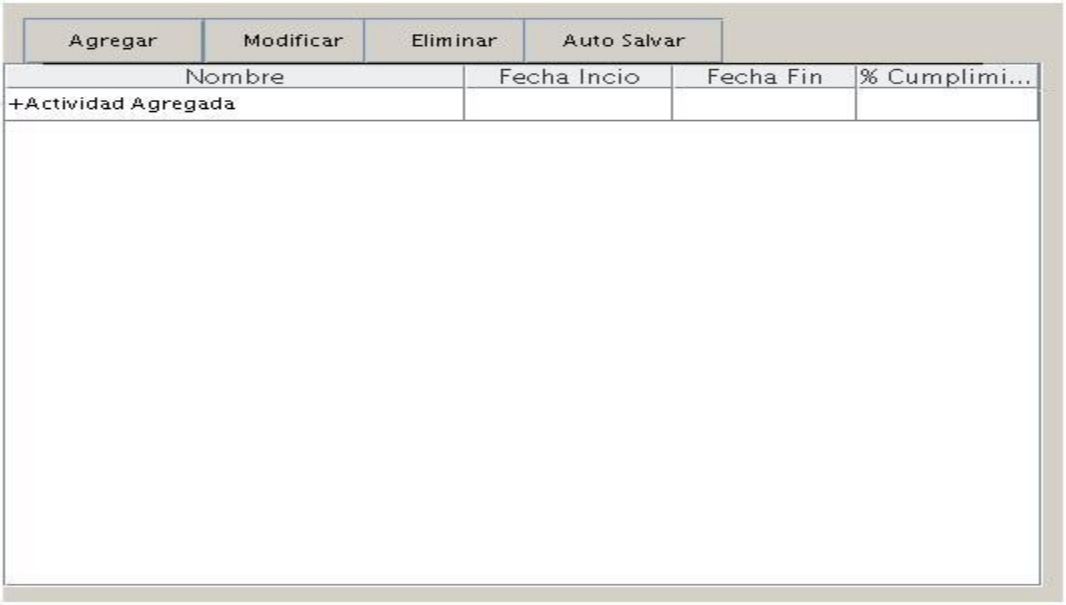


Figura 11 : Prototipo de interfaz para requisitos R6, R7, R8.

2.8.4 Prototipo interfaz para Modificar una actividad

El sistema le permite al usuario modificar una actividad a través de otra interfaz y además se le asigna una categoría a la actividad. Posee además un vínculo para adicionar una nueva categoría. Este prototipo recoge los requisitos funcionales R9 Mostrar información de una actividad y R11 Asignar categoría a una actividad. Ella se modifica cuando des clic en el botón Aceptar.

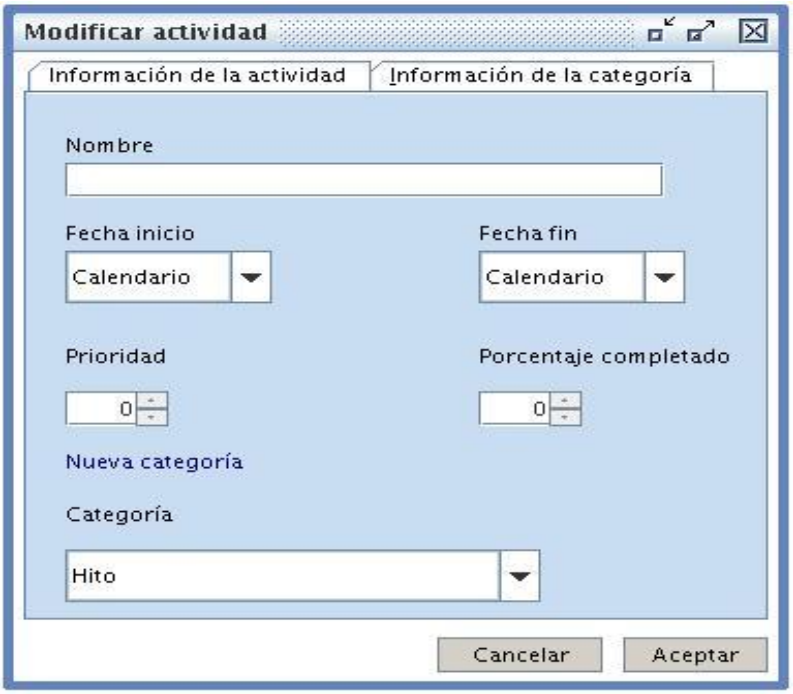


Figura 12: Prototipo interfaz para los requisitos R8, R9 y R11.

2.8.5 Prototipo interfaz para Modificar el valor de la categoría

El sistema permite mostrar el nombre y el valor de la categoría, además de que el usuario puede modificar esos valores y salvarlos en la base de datos. También cuenta con un vínculo para gestionar los campos de la categoría. En conjunto con el requisito funcional R8 Modificar actividad es que se modifica la actividad, asignándole una categoría con su valor.

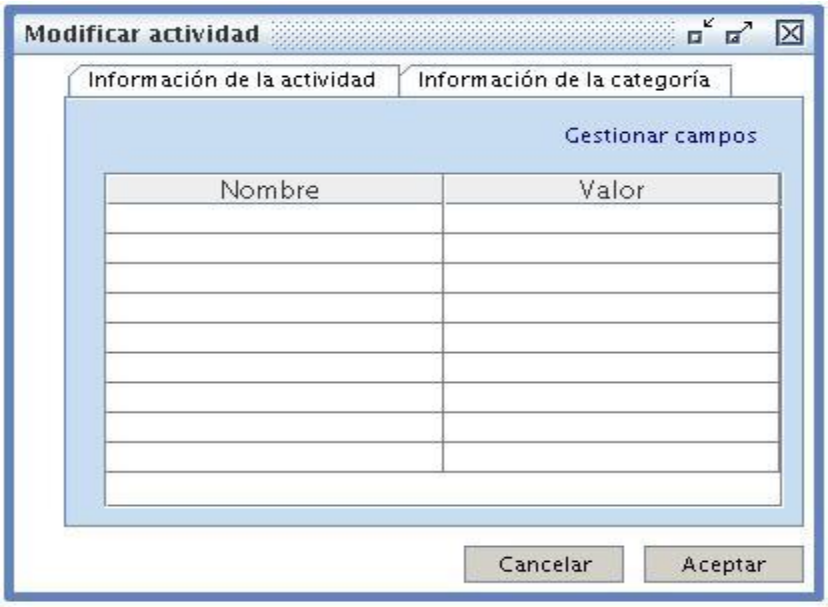


Figura 13: Prototipo interfaz para el requisito funcional R15.

2.8.6 Prototipo interfaz para Adicionar una nueva categoría

El sistema debe permitir adicionar una nueva categoría al conjunto existente para ser asignada a una actividad. Este prototipo recoge los requisitos funcionales R12 Adicionar campo, personalizado a una categoría y R13 Eliminar campo personalizado a una categoría.

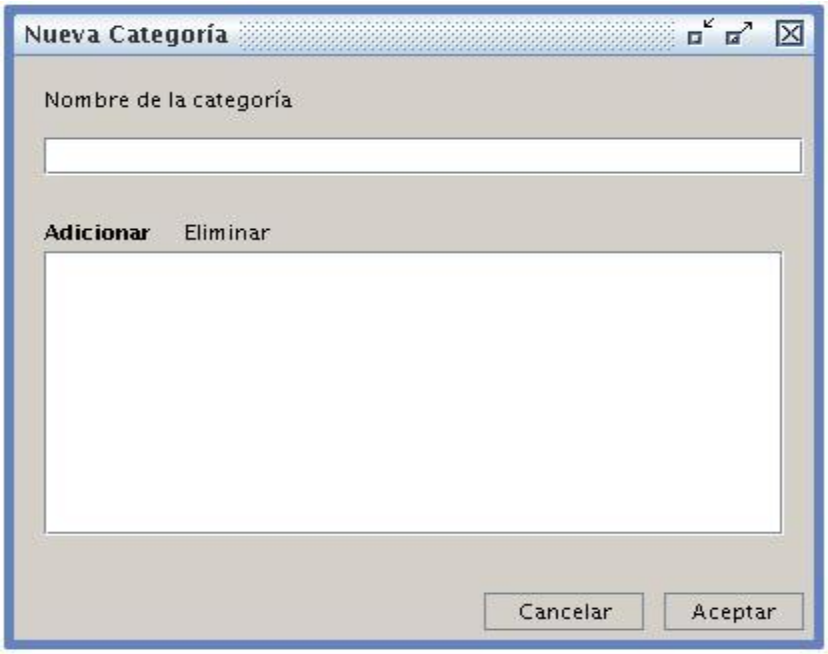


Figura 14 : Prototipo interfaz para el requisito funcional R10, R12 y R13.

2.8.7 Prototipo interfaz para Modificar categoría

El sistema permite mostrar información de la categoría, modificar la categoría así como adicionar y eliminar campos, además le puedes cambiar el nombre, todo se modificará cuando des clic en el botón Aceptar. El prototipo recoge el requisito funcional R17 Mostrar información de la categoría.

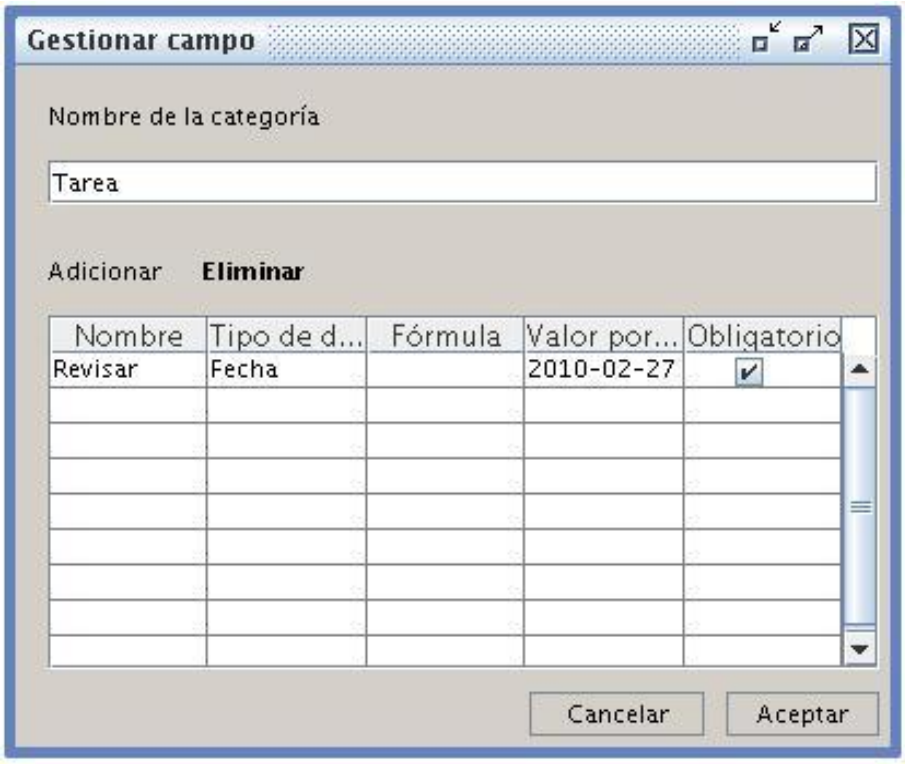


Figura 15: Prototipo interfaz para los requisitos funcionales R16 y R17.

2.9 Conclusiones

En el presente capítulo se tuvo como punto de partida el diagrama de casos de uso del sistema, así como los requisitos funcionales y no funcionales. Se expusieron aspectos referentes al proceso del diseño, se plasmaron los diagramas de clases y de interacción a utilizar para la implementación y se reflejaron los distintos prototipos de interfaz.

Capítulo 3 Implementación y Pruebas.

3.1 Introducción

En el presente capítulo serán abordados todos los temas referentes a la implementación y pruebas del sistema. En el mismo se muestra el diagrama de componentes y los estándares de codificación utilizados. Además, se especifican los casos de pruebas de unidad realizadas al software y las métricas de diseño.

3.2 Implementación

La implementación es el principal flujo de trabajo en la fase de construcción. Describe cómo los elementos del modelo de diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo con los nodos específicos en el modelo de despliegue. Está determinado por el lenguaje de programación y tiene como objetivo llevar a cabo la implementación de cada una de las clases significativas del diseño. (22)

En este flujo de trabajo se define la organización del código en términos de los subsistemas de implementación organizados en capas. Se implementan los elementos del diseño en términos de implementación, obteniéndose los componentes necesarios para el funcionamiento de la aplicación así como el diagrama de componentes, que conforma lo que se conoce como un modelo de implementación al describir los componentes a construir, su organización y dependencia entre nodos físicos en los que funcionará la aplicación.

3.3 Diagrama de componentes

Este diagrama muestra el conjunto de componentes y sus relaciones. Permite describir la vista de implementación estática de un sistema. Un componente se corresponde con una o más clases, interfaces o colaboraciones. El siguiente diagrama muestra los componentes del módulo siguiendo la arquitectura Modelo Vista Controlador.

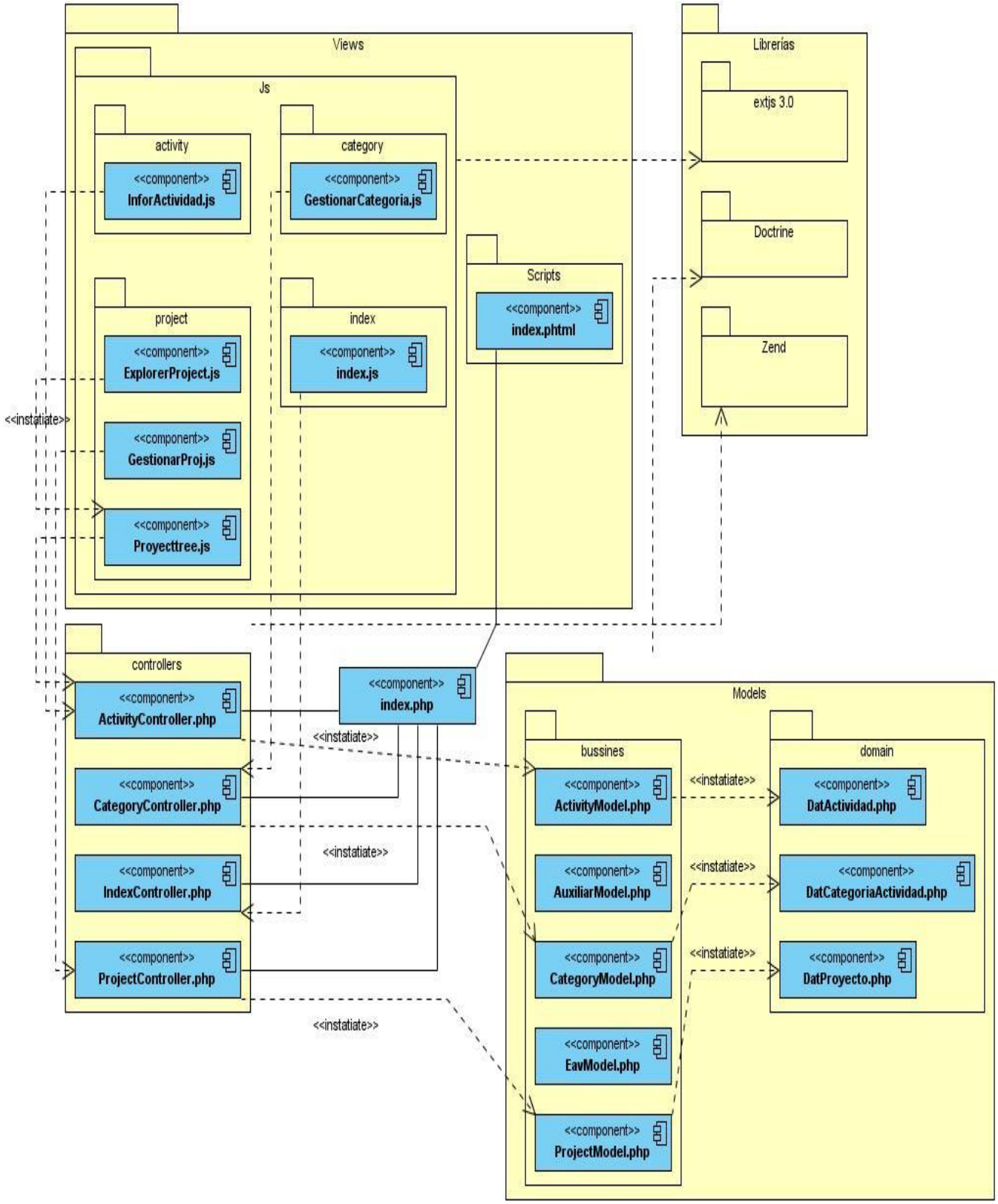


Figura 16: Diagrama de componentes

3.4 Estándares de codificación utilizados

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Un estándar de programación define la nomenclatura de las variables, objetos, métodos y funciones, teniendo que ver además, con el orden y legibilidad del código escrito.

3.4.1 Nomenclatura de las clases

Los nombres de las clases comienzan con la primera letra con mayúscula y el resto con minúscula, en caso de ser una palabra compuesta se empleará la notación *PascalCasing*[2].

Ejemplo:

GestionarCategoría.

3.4.2 Nomenclaturas según el tipo de clases

Clases controladoras:

Llevan después del nombre la palabra “Controller”.

Ejemplo:

CategoryController.

Clases de los modelos:

Se encuentran dentro de Business (negocio) y llevan la palabra “Model”.

Ejemplo:

CategoryModel.

Las clases que se encuentran dentro de Domain el nombre que reciben es el de la tabla en la base de datos.

Ejemplo:

DatCategoría.

[2] Es como la notación húngara pero sin prefijos. En este caso, los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula. (25)

3.4.3 Nomenclatura de las funciones

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación *CamelCasing*[3].

Ejemplo:

gestionarCategoria().

En el caso de las funciones en las clases controladoras se especifica el nombre de la acción, todo en minúscula y seguido el sufijo"Action".

Ejemplo:

categoryAction().

3.4.4 Nomenclatura de las variables

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

Ejemplo:

$categoriaNombre.

3.4.5 Normas de comentarios

Es necesario comentar todo lo que se haga dentro del desarrollo, es decir, establecer las pautas que conlleven a lograr un código más legible y reutilizable, de manera que se pueda aumentar el mantenimiento a lo largo del tiempo.

Nomenclatura de los comentarios.

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

Entre las clases:

Antes de la declaración de una clase se escribe una breve descripción donde se explica el propósito de la misma. A continuación se muestra un ejemplo:

```
/*
 * GestionarCategoria
 * @author: Yuniesky Armenteros
```

[3] Es parecido al Pascal-Casing con la excepción que la letra inicial del identificador no debe estar en mayúscula. (25)

* @author: Abelardo López
* @package *SINTEL
* @copyright *
* @version 1.0
*/

En las funciones:

Antes de la declaración de la función se escribe una breve descripción donde se explica el propósito de la misma.

Ejemplo:

//Operación que devuelve una categoría.

3.5 Tratamientos de errores

Para garantizar el correcto funcionamiento del sistema se debe tener en cuenta un tratamiento de errores, el sistema captura todas aquellas excepciones que son lanzadas y se le facilita un tratamiento para que no colapse.

En el sistema se identifican cada uno de los puntos en los que puede dar el error, los mismos son capturados y gestionados en los métodos implementados, en el cual se definen los posibles errores y cómo deben ser mostrados los mismos de manera entendible para el usuario, además este manejo de errores se realiza en dependencia del tipo de error lanzado. Se utilizó principalmente el Try {} Catch (Exception e) {}.

Otros elementos generales del tratamiento de errores son los siguientes:

- ✓ La corrección de errores en la introducción de datos será clara y fácil de realizar.
- ✓ La entrada de datos incorrecta será detectada claramente e informada al usuario.
- ✓ Todos los textos y mensajes en pantalla aparecerán en idioma español.

3.6 Descripción de las principales clases a utilizar

3.6.1 Clases controladoras

Estas clases tienen como objetivo controlar el flujo de datos de la aplicación, representan coordinación, secuencia, transacciones y control de objetos, se usan para encapsular el manejo de un caso de uso en concreto, coordinando las actividades de los objetos que implementan la funcionalidad del mismo, por lo que definen el flujo de datos y las transacciones dentro de un caso de uso delegando el trabajo a otros objetos.

A continuación se describen algunas de las clases más importantes:

ActivityController: es la que se dedica a gestionar las actividades que posee un proyecto.

Nombre: ActivityController	
Tipo de clase: Controladora	
Atributo	Tipo
Para cada responsabilidad	
Nombre	Descripción:
init()	Constructor de la clase
activityAction()	Devuelve la información del la actividad
activitiesAction()	Devuelve una lista de todas las actividades
nameAction()	Devuelve el nombre de una actividad
namesAction()	Devuelve una lista de todos los nombres de las actividades
addAction()	Adiciona una actividad
deleteAction()	Elimina una actividad
modifyAction()	Modifica una actividad
newparentAction()	Cambiar padre a una actividad
childrenAction()	Devuelve la lista de hijos de una actividad

Tabla 1: Clase controladora ActivityController

CategoryController: es la clase controladora donde se gestiona las categorías que posee una actividad, además de que se le agregan nuevos campos a la categoría.

Nombre: CategoryController	
Tipo de clase: Controladora	
Atributo	Tipo
Para cada responsabilidad	
Nombre	Descripción:
init()	Constructor de la clase
categoryAction()	Devuelve la información de la categoría
categoriesAction()	Devuelve una lista de todas las categorías
nameAction()	Devuelve el nombre de una categoría
namesAction()	Devuelve todos los nombres de las categorías

addAction()	Adiciona una categoría
deleteAction()	Elimina una categoría
modifyfieldAction()	Modifica los campos de una categoría
fieldsAction()	Devuelve todos los campos de una categoría
addfieldAction()	Adiciona campos a una categoría
deletefieldAction()	Elimina los campos de la categoría
modifyAction()	Modifica una categoría
typefieldAction()	Devuelve el tipo de dato de una categoría

Tabla 2: Clase controladora CategoryController

ProjectController: se gestionan todos los proyectos que posee el ministerio.

Nombre: ProjectController	
Tipo de clase: Controladora	
Atributo	Tipo
Para cada responsabilidad	
Nombre	Descripción:
init()	Constructor de la clase
projectAction()	Devuelve la información de un proyecto
projectsAction()	Devuelve una lista de todos los proyectos
nameAction()	Devuelve el nombre de un proyecto
namesAction()	Devuelve todos los nombres de los proyectos
addAction()	Adiciona un proyecto
deleteAction()	Elimina un proyecto
modifyAction()	Modifica un proyecto
childrenAction()	Devuelve los hijos de un proyecto
statesAction()	Determina el estado de un proyecto
typesAction()	Determina el tipo de un proyecto

Tabla 3: Clase controladora ProjectController

3.7 Casos de pruebas

La realización de los casos de pruebas tiene como objetivo demostrar al cliente la reacción que corresponderá por parte del sistema luego de realizar alguna acción en el mismo. Para un mejor entendimiento de las respuestas o posibles funcionalidades que brindará el sistema, según la necesidad del usuario. La descripción de los casos de prueba del sistema se realizó por escenarios y casos de uso.

Las mismas son actividades en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. (23)

3.7.1 Pruebas de Unidad

Es la prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control de datos están cubiertos, funcionando como se espera. La prueba de unidad siempre está orientada a caja blanca, aunque para realizar las mismas es necesario probar el flujo de datos desde la interfaz del componente. Si los datos no se comportan correctamente al ser introducidos en la aplicación, todas las demás pruebas no cumplen función hacerla.

3.7.2 Pruebas de caja negra

Las pruebas de caja negra son las que se llevan a cabo sobre la interfaz del software, o sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada, obteniéndose un resultado correcto y la integridad de la información externa se mantiene. (24)

Estas pruebas permiten encontrar:

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.

Algunas técnicas utilizadas en la prueba de caja negra son:

✓ **Partición de Equivalencia:** Técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición equivalente se dirige a que descubran las clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

Ejemplo: Si una condición de entrada especifica un rango de valores, o sea, si un contador puede ir de 1 a 999, la clase válida sería "1 <= contador <= 999. Mientras que las clases no válidas serían "contador < 1" y "contador > 999".

✓ **Análisis de Valores Límite*:*** La experiencia muestra que los casos de prueba que exploran las condiciones límite producen mejor resultado que aquellos que no lo hacen. Las condiciones límite son aquellas que se hallan en los márgenes de la clase de equivalencia, tanto de entrada como de salida. Por ello, se ha desarrollado el análisis de valores límite como técnica de prueba. Esta técnica nos lleva a elegir los casos de prueba que ejerciten los valores límite.

Las pautas para desarrollar esta técnica es:

− Si una condición de entrada especifica un rango de valores, se diseñarán casos de prueba para los dos límites del rango, y otros dos casos para situaciones justas por debajo y por encima de los extremos.

− Si una condición de entrada especifica un número de valores, se diseñan dos tipos de pruebas para los valores mínimo y máximo, además de otros dos para valores justos por encima del máximo y justo por debajo del mínimo.

− Aplicar las reglas anteriores a los datos de salida.

− Si la entrada o salida de un programa es un conjunto ordenado, habrá que prestar atención a los elementos primero y último del conjunto.

✓ **Grafos de causa-efecto:** Es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

A continuación, se muestra un ejemplo de esta prueba a una interfaz del software que pertenece al caso de uso Gestionar proyecto:

Interfaz de adicionar proyecto.

Caso de uso: Gestionar proyecto.

Caso de prueba: Entrada de datos a la interfaz para adicionar un nuevo proyecto.

Entrada: Al abrir la ventana de nuevo proyecto, se le introducen los datos: nombre del proyecto, fecha que va a durar el mismo, es decir, fecha inicio y fin, estado del proyecto, tipo de proyecto y enunciado del proyecto. Casi todos los campos son obligatorios, excepto el enunciado, lo que obliga al usuario a no dejar en blanco estos campos.

Resultado: Se adiciona el proyecto en la base de datos.

Condiciones: Los campos obligatorios deben llenarse para poder adicionar el proyecto, además el nombre debe empezar con letras. También hay que tener en cuenta que el proyecto no exista, si no tampoco se adicionaría.

3.7.3 Pruebas de caja blanca

En las pruebas de la caja blanca del software se comprueban los caminos lógicos del software proponiendo casos de prueba en los que se ejerciten conjuntos específicos de condiciones o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado.

Tipos de prueba de caja blanca:

Prueba de Condición: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.

Prueba de Flujo de Datos: Se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

Prueba de Bucles: Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

Prueba del Camino Básico: Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes, por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. (24)

La técnica del camino básico permite obtener una medida de la complejidad lógica del código de cada método, programa o módulo dado. Es además una de las más eficientes en cuanto a cobertura de código, pues logra que se ejecuten todos los bucles en sus límites operacionales.

Los pasos que se siguen para aplicar esta técnica son:

- ✓ A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- ✓ Se determina un conjunto básico de caminos independientes.
- ✓ Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.
- ✓ Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

✓ Para aplicar la técnica del camino básico se debe introducir la notación para la representación del flujo de control, representado por un grafo de flujo en el cual:

✓ Cada nodo del grafo corresponde a una o más sentencias de código fuente.

✓ Todo segmento de código de cualquier programa se puede traducir a un grafo de flujo.

✓ Se calcula la complejidad ciclomática del grafo.

Para la realización del grafo hay tres componentes fundamentales:

Nodo: son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Aristas: son constituidas por las flechas del grafo, iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aún cuando el nodo no representa la sentencia de un procedimiento.

Regiones: son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

Para realizar la prueba de caja blanca, específicamente la prueba del camino básico, es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar.

A continuación se expone un ejemplo de cómo se aplicó este tipo de prueba a un método del software; inicialmente se enumeran las sentencias del código escogido.

```
public function addAction() {
    $nombreproyecto = $this->_request->getPost ( 'nombreproyecto' );1
    $idtipoproyecto = $this->_request->getPost ( 'tipoproyectoid' );1
    $idestadoproyecto = $this->_request->getPost ( 'estadoproyectoid' );1
    $fechainicio = $this->_request->getPost ( 'fechainicio' );1
    $fechafin = $this->_request->getPost ( 'fechafin' );1
    $alcance = $this->_request->getPost ( 'alcance' );1
    $parent = $this->_request->getPost ( '_parent' );1
    if ($parent == "")2
        $parent = null;3
    try {4
        $proyecto = ProjectModel::add ( $nombreproyecto,4
        $idtipoproyecto, $idestadoproyecto, $fechainicio, $fechafin, $alcance, $parent );4
        echo "{'success':true, 'datos':" . json_encode ( $proyecto ) . "}";4
    } catch ( Exception $e ) {4
        $msg = $e->getMessage ();5
        echo "{'success':false, 'message':$msg}";}5
}6
```

Figura 17: Representación del algoritmo adicionar proyecto

Seguidamente se construye el grafo asociado:

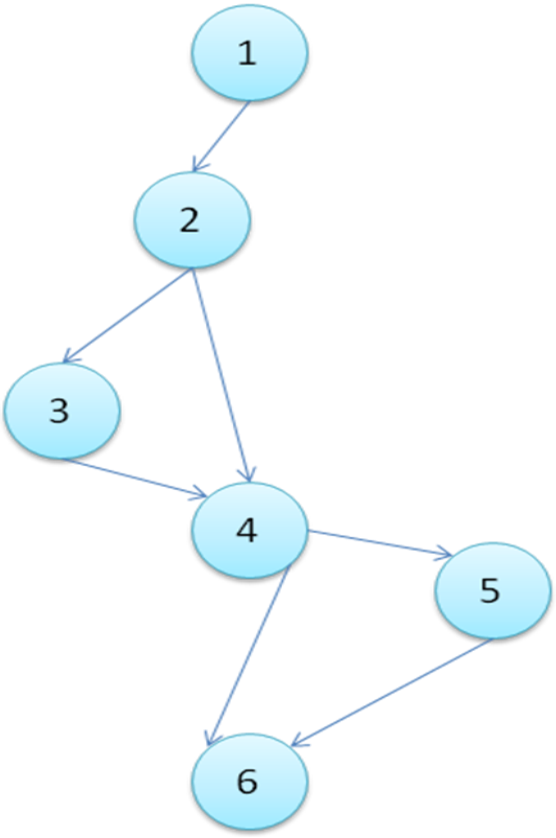


Figura 18 : Grafo de flujo asociado al algoritmo adicionar proyecto

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante tres fórmulas, las cuales tienen que mostrar el mismo resultado para asegurar que el cálculo de la complejidad es correcto.

Fórmulas para calcular complejidad ciclomática:

1. V (G) = (A – N) + 2.

2. V (G) = P + 1.

3. V (G) = R.

Aplicando estas fórmulas al grafo de flujo de la figura 18 se obtienen los siguientes resultados:

Calculando mediante la fórmula 1:

V (G) = (A – N) + 2

V (G) = (7 – 6) + 2

V (G) = 3

Siendo “A” las aristas y “N” los nodos.

Calculando mediante la fórmula 2:

V (G) = P + 1

V (G) = 2 + 1

V (G) = 3

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

Calculando mediante la fórmula 3:

V (G) = R

V (G) = 3

Siendo “R” la cantidad total de regiones, para cada fórmula “V (G)” representa el valor del cálculo.

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es 3, lo que significa que existen tres posibles vías por donde el flujo puede circular. El valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Un camino independiente es cualquier camino del

programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El mismo, debe moverse por lo menos por una arista no recorrida anteriormente.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo:

Camino básico #1:

1 – 2 – 3 – 4 – 5 – 6

Camino básico #2:

1 – 2 – 3 – 4 – 6

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para el procedimiento, se debe realizar al menos un caso de prueba por cada camino básico.

Para realizarlos es necesario cumplir con las siguientes exigencias:

✓ Descripción: Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no sé entre algún dato erróneo.

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

✓ Entrada: Se muestran los parámetros que entran al procedimiento.

✓ Resultados esperados: Se expone el resultado que el procedimiento espera.

Caso de prueba para el camino básico #1:

Camino 1: [1 – 2 – 3 – 4 – 5 – 6]

Descripción:

Los datos de entrada cumplirán con los siguientes requisitos:

Los parámetros $nombreProyecto, $idtipoProyecto, $idestadoProyecto, $fechaInicio, $fechaFin, tendrán valores vacíos, o al menos uno es vacío.

Condición de ejecución:

El nombre del proyecto será DATEC.

El tipo del proyecto tendrá valor vacío.

El estado del proyecto será activo.

El proyecto tendrá una fecha inicio y fin.

Entrada:

$nombreProyecto = ‘DATEC’

$idtipoProyecto = “ ”

$idestadoProyecto = Activo

$fechaInicio = 2010-02-28

$fechaFin = 2010-09-26

Resultados esperados:

Se espera que este proyecto no se adicione ya que no se especificó el tipo de proyecto que es.

Caso de prueba para el camino básico #2:

Camino 2 [1 – 2 – 3 – 4 – 6]

Descripción:

Los datos de entrada cumplirán con los siguientes requisitos:

Los parámetros $nombreProyecto, $idtipoProyecto, $idestadoProyecto, $fechaInicio, $fechaFin, todos tendrán un valor.

Condición de ejecución:

El nombre del proyecto será DATEC.

El tipo del proyecto tendrá valor desarrollo.

El estado del proyecto será activo.

El proyecto tendrá una fecha inicio y fin.

Entrada:

$nombreProyecto = 'DATEC'

$idtipoProyecto = Desarrollo

$idestadoProyecto = Activo

$fechaInicio = 2010-02-28

$fechaFin = 2010-09-26

Resultados esperados:

Se espera que este proyecto se adicione, ya que el proyecto no está dentro de otro y muestre el mensaje: proyecto salvado satisfactoriamente.

Luego de aplicar los distintos casos de pruebas, se pudo comprobar que el flujo de trabajo de la función está correcto, pues cumple con las condiciones necesarias que se habían planteado.

3.8 Métricas de software

Las métricas son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo de software y proyectos.

Objetivos de las métricas de diseño:

- ✓ Llevar un control de la calidad del producto que se está desarrollando.
- ✓ Estimar el impacto que ese producto tendrá en las fases posteriores del proceso de desarrollo.

✓ Ayudan a que el diseño evolucione a un nivel superior de calidad.

Un aspecto importante a tener en cuenta en la evaluación de la calidad del diseño ha sido la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objeto referenciadas por Pressman, teniendo en cuenta que este estudio brinda un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software.

Atributos de calidad que se abarcan:

✓ **Responsabilidad (Cohesión)**: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

✓ **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 costos y la planificación del proyecto.

✓ **Complejidad de implementación**: Consiste en el grado de dificultad que tiene que 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.

✓ **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) diseñado. (26)

3.8.1 Tamaño Operacional de Clase

Está dado por el número de métodos asignados a una clase. Los atributos que afecta son la Responsabilidad, Complejidad de implementación y la Reutilización, de manera que mientras mayor sea el Tamaño Operacional de Clase (TOC) mayor será la Responsabilidad y Complejidad de implementación, mientras que su Reutilización disminuye. (26)

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 4: Tamaño Operacional de Clases (TOC)

Resultados del instrumento de evaluación del TOC

Ver instrumentos y tabla de resultados en (Anexo 3 Instrumento de medición de la métrica TOC).

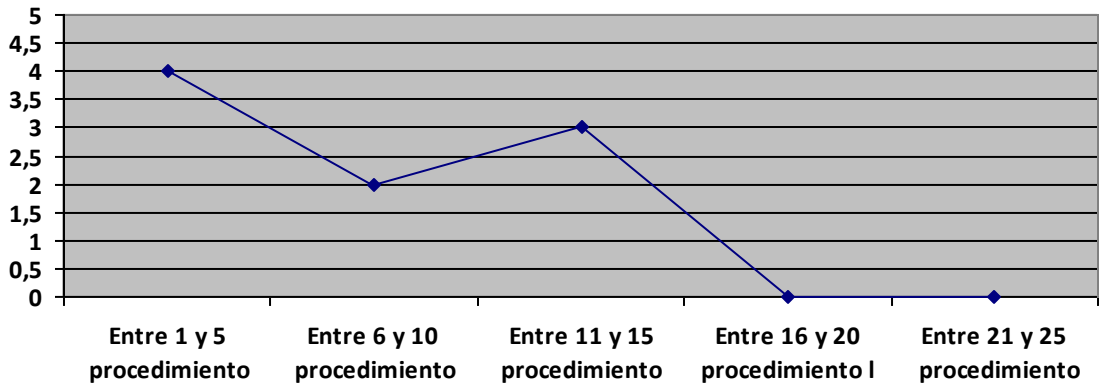


Figura 19: Resultados obtenidos en el instrumento agrupados en los intervalos definidos.

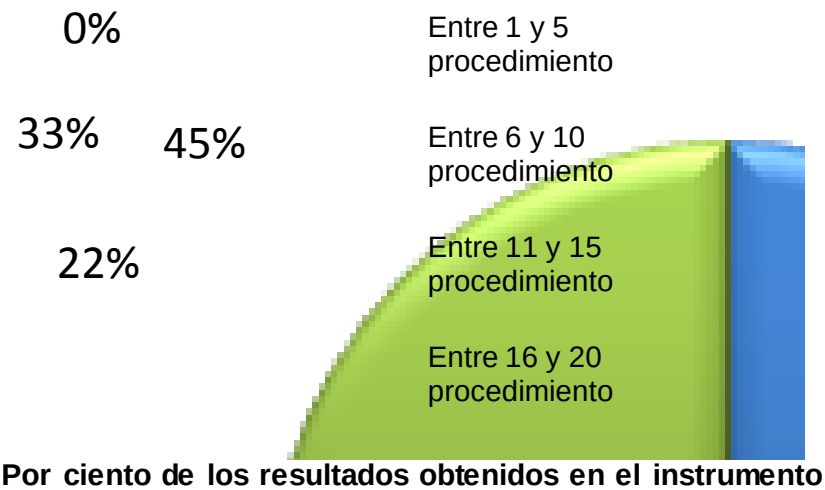


Figura 20: Por ciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

Figura 21: Incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

Figura 22: Incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

Figura 23: Incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño del módulo Gestión de indicadores tiene una calidad aceptable teniendo en cuenta que el 89% de las clases que posee el producto tiene menos cantidad de operaciones que la mitad del valor máximo registrado en las mediciones. Además, el 89% de las clases poseen evaluaciones positivas en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).

3.8.2 Relaciones entre clases

Está dado por el número de relaciones de uso de una clase. Los atributos que afecta son el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de pruebas. De manera que mientras mayor sean las Relaciones entre Clases (RC) mayor será el Acoplamiento, la Complejidad de mantenimiento y la Cantidad de pruebas, mientras que su Reutilización disminuye. (26)

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 5: Relaciones entre clases (RC)

Resultados del instrumento de evaluación de la métrica RC

Ver instrumentos y tabla de resultados en (Anexo 4 Instrumento de medición de la métrica RC).

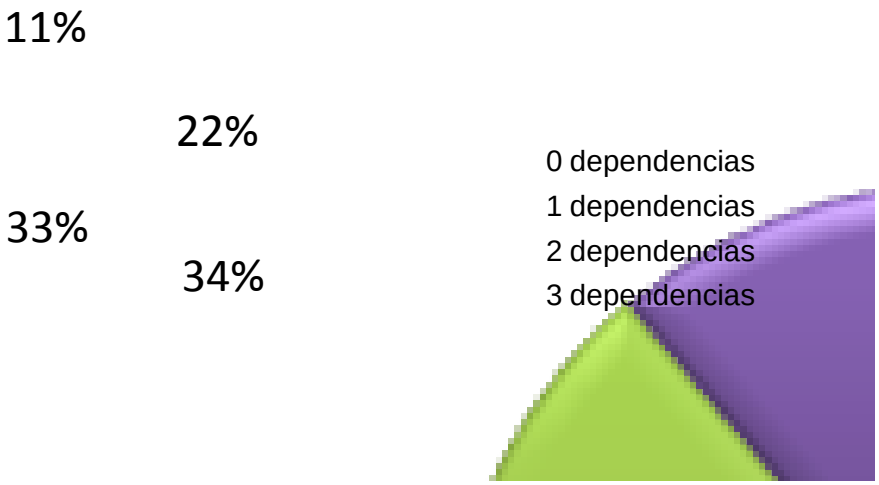


Figura 24: Por ciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

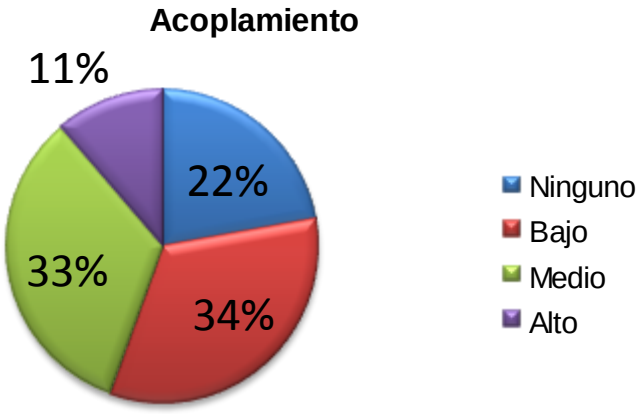


Figura 25: Incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

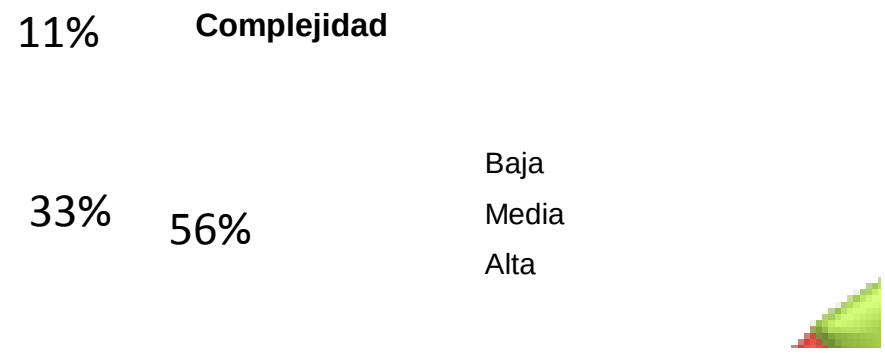


Figura 26: Incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

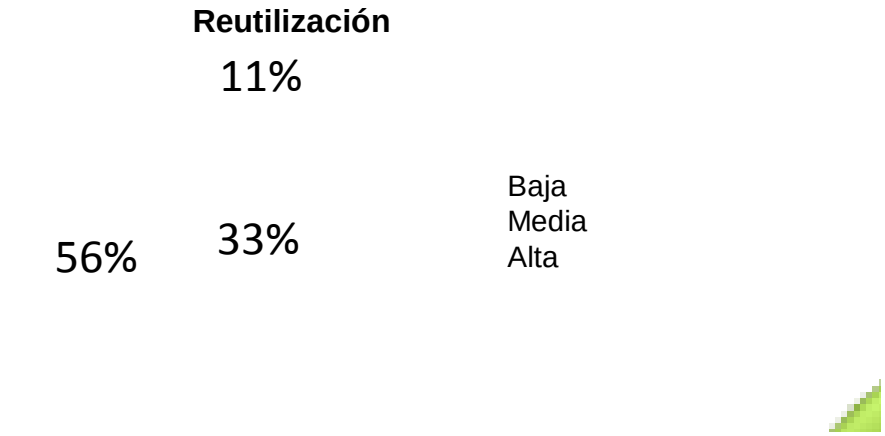


Figura 27: Incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

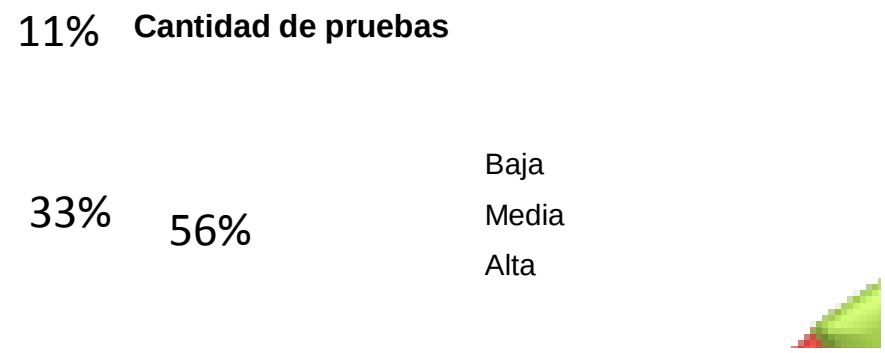


Figura 28: Incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

Analizando los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del módulo tiene una calidad aceptable teniendo en cuenta que el 89% de las clases poseen menos de tres dependencias de otras clases. Además, el 89% posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 89% de las clases.

A manera de resumen se han tabulado los resultados obtenidos en la siguiente tabla.

Atributos	TOC (en %)	RC (en %)
Responsabilidad	89	
Complejidad de implementación	89	
Reutilización	89	89
Acoplamiento		88
Complejidad del mantenimiento		89
Cantidad de pruebas		89

Tabla 6: Resultados generales obtenidos por las métricas.

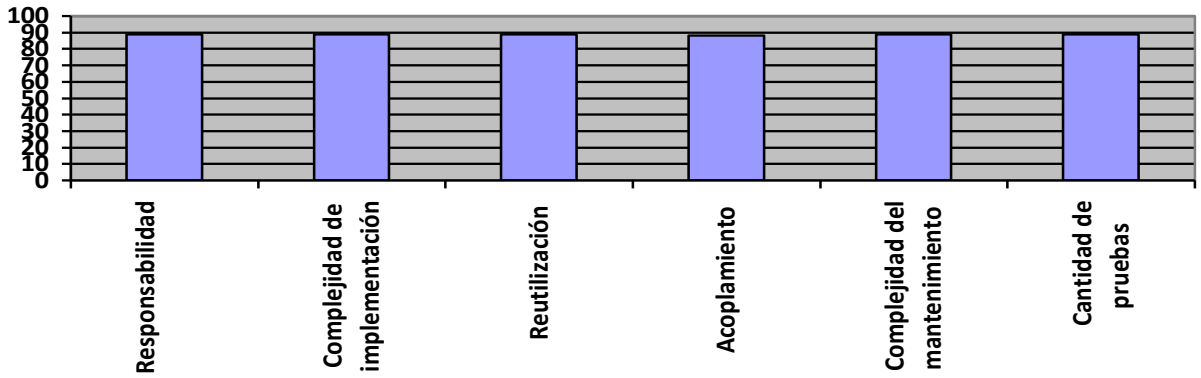


Figura 29: Gráfica de los resultados obtenidos de los atributos de calidad evaluados en las métricas.

3.9 Conclusiones

En el capítulo se ha realizado la descripción de la implementación, se muestra el diagrama de componentes, el cual contiene paquetes que representan las integraciones que existen dentro del sistema desarrollado. Además, se le realizaron las pruebas de unidad, principalmente las de caja blanca, demostrando la calidad y eficiencia del módulo.

Conclusiones generales

Como resultado del trabajo:

✓ El uso del marco CEDRUX facilitó la implementación y garantizó la aplicación de buenas prácticas de desarrollo.

✓ La selección y utilización adecuada de patrones potenciaron un diseño correcto, esencial para una posterior implementación.

✓ Con el diseño e implementación del módulo de Gestión de indicadores para el proyecto del Ministerio del Poder Popular de la Comunicación e Información de Venezuela, se logró la automatización de los objetivos del mismo.

Recomendaciones

Luego de haber cumplimentado los objetivos propuestos mediante la realización del trabajo, se recomienda que:

- ✓ Los componentes del módulo pasen a formar parte del repositorio de componentes del Centro.
- ✓ Identificar qué proyectos similares podrían reutilizar los componentes y la solución.

Referencias Bibliográficas

1. Sistemas de Gestión. [En línea] [Citado el: 25 de enero de 2010.] http://www.softwareprojects.org/free-project.

2. Metodología. [En línea] [Citado el: 26 de enero de 2010.] Documentos de Metodología de Desarrollo Líneas de Productos del proyecto.

3. Patrones. [En línea] [Citado el: 2 de febrero de 2010.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

4. Programación orientada a objetos (POO). [En línea] [Citado el: 4 de febrero de 2010.] http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/I_1.htm..

5. PHP. [En línea] [Citado el: 3 de febrero de 2010.] http://www.sitgeshost.com/cms-wordpress-joomla-php-seo-dominios-hopedaje-sitges/php-por-encargo.

6. XHTML. [En línea] [Citado el: 5 de febrero de 2010.] http://www.w3c.es/Divulgacion/Guiasbreves/XHTML.

7. CSS. Libro CSS Introducción. [Citado 26 de enero del 2010]. .

8. Introducción a JSON en JavaScript y .Net. [En línea] [Citado el: 1 de febrero de 2010.] http://hanzcocchi.net/introduccion-a-json-en-javascript-y-net/..

9. XML. [En línea] [Citado el: 8 de febrero de 2010.] http://www.w3.org/XML.

10. AJAX. [En línea] [Citado el: 9 de febrero de 2010.] http://www.librosweb.es/ajax/capitulo1.html.

11. ExtJS. [En línea] [Citado el: 10 de febrero de 2010.] http://www.extjs.com/.

12. Doctrine. [En línea] [Citado el: 10 de febrero de 2010.] http://www.tecnoretales.com/programacion/que-es-doctrine-orm.

13. Visual Paradigm para UML. [En línea] [Citado el: 11 de febrero de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%5Bcuenta_de_Plataforma_de_Java_14715_p.

14. PgAdmin III. [En línea] [Citado el: 13 de febrero de 2010.] http://guia-ubuntu.org/index.php?title=PgAdmin_III.

15. Subversion. [En línea] [Citado el: 12 de febrero de 2010.] http://www.guia-ubuntu.org/index.php?title=SVN.

16. Framework . [En línea] [Citado el: 14 de febrero de 2010.] http://www.acm.org/crossroads/espanol/xrds7-4/frameworks.html.

17. Leopoldo Magaña, Carlos. Carlos. Zend Framework. [En línea] [Citado el: 15 de febrero de 2010.] http://www.carlosleopoldo.com/post/zend-framework-una-introduccion/.

18. Doctrine Framework . [En línea] [Citado el: 16 de febrero de 2010.] http://www.doctrine-project.org/.

19. Arquitectura. [En línea] [Citado el: 19 de febrero de 2010.] http://www.desarrolloweb.com.

20. Ingeniería de Software II. Conferencia 1. [En línea] [Citado el: 27 de febrero de 2010.]

21. Ingeniería de Software I . Conferencia #7 Flujo de Trabajo Análisis y Diseño. [En línea] [Citado el: 27 de febrero de 2010.]

22. Ingeniería de Software II. Conferencia_Pruebas. [En línea] [Citado el: 4 de marzo de 2010.] http://eva.uci.cu/mod/resource/view.php?id=14103 .

23. Ingeniería de Software I Conferencia # 5 Flujo de Trabajo Implementación. [En línea] [Citado el: 3 de marzo de 2010.] http://eva.uci.cu/.

24. Ingeniería de Software I Conferencia # 7 Flujo de Trabajo Prueba. [En línea] [Citado el: 5 de marzo de 2010.] HYPERLINK "http://eva.uci.cu/" http://eva.uci.cu/

25. Coello Costa, Helkyn R. informatízate. [Citado: 4 de marzo, 2010.] http://www.informatizate.net/articulos/dime_como_programas_y_te_dire_quien_eres_23082004.html.

26. Roger S. Pressman. Ingeniería de software. Un enfoque práctico.[En línea] [Citado el: 6 de abril de 2010]. http://biblioteca.uci.cu.

Bibliografía

1. Conferencia 7 Prueba. [En línea] [Citado el: 10 de marzo de 2010.] http://eva.uci.cu.

2. Ext 3.0 - API Documentation. [En línea] [Citado el: 15 de enero de 2010.] http://www.extjs.com.

3. Herramientas CASE. [En línea] [Citado el: 17 de enero de 2010.] http://www.cyta.com.ar/biblioteca/bdlibros/proyectoinformatico/libro/c5/c5.htm.

4. Ingeniería de Software I Conferencia # 5 Flujo de Trabajo Implementación. [En línea] [Citado el: 25 de enero de 2010.] http://eva.uci.cu.

5. Introducción a JavaScript. [En línea] [Citado el: 6 de diciembre de 2009.] http://www.librosweb.es.

6. Introducción a JSON. [En línea] [Citado el: 8 de marzo de 2010.] http://www.json.org/json-es.html.

7. Pérez, Javier Eguíluz. CSS Avanzado. [En línea] [Citado el: 5 de diciembre de 2009.] http://www.librosweb.es.

8. Larman, C. Uml y patrones: Introducción al análisis y programación orientada a objetos. [En línea] [Citado el: 20 de enero de 2010.] http://biblioteca.uci.cu..

9. Magaña, Leopoldo. ZendFramework. [En línea] [Citado el: 16 de enero de 2010.]

10. Lussón, Juan Carlos Quevedo. Metodología. [En línea] [Citado el: 2 de febrero de 2010.]

11. Manual Doctrine. [En línea] [Citado el: 2010 de febrero de 2010.]

12. Mendoza Navarro, Javier. Diseño de sistemas. [En línea] [Citado el: 16 de enero de 2010.]

13. Patrones de diseño. [En línea] [Citado el: 6 de febrero de 2010.] http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-12.pdf..

14. PgAdmin III. [En línea] [Citado el: 6 de febrero de 2010.] http://guia-ubuntu.org/index.php?title=PgAdmin_III.

15. PHP. [En línea] [Citado el: 7 de marzo de 2010.] http://www.php.net/..

16. Pressman, Roger S. Ingeniería de software. Un enfoque práctico. [En línea] [Citado el: 24 de marzo de 2010.]

17. Frederick, Shea. Learning Ext JS. [En línea] [Citado el: 12 de diciembre de 2010.]

18. Spket IDE. [En línea] [Citado el: 6 de marzo de 2010.] http://www.spket.com.

Glosario de términos

Algoritmo: Es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.

Atributo: Contenedor de un tipo de datos asociados a un objeto, que hace los datos visibles desde fuera del objeto, y cuyo valor puede ser alterado por la ejecución de algún método.

Clase: Definiciones de las propiedades y comportamiento de un conjunto de objetos concretos. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

Componente: El componente es la unidad de construcción elemental del diseño físico. Las características de un componente son:

- ✓ Se define según como interactúa con otros.
- ✓ Encapsula sus funciones y sus datos.
- ✓ Es reutilizable a través de las aplicaciones.
- ✓ Puede verse como una caja negra.
- ✓ Puede contener otros componentes.

Evento: Un suceso en el sistema. El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.

Framework: Conjunto de interfaz de programación de aplicaciones (APIs) y herramientas destinadas a la construcción de un determinado tipo de aplicaciones de manera generalista.

Implementación: Proceso por el cual se escribe en un lenguaje de programación, se prueba, se depura y se mantiene el código fuente de un programa informático.

Mensaje: Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

Método: Algoritmo asociado a un objeto, cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

Módulo: A efectos prácticos, hablar de programas es lo mismo que hablar de productos de software o hablar de módulos de software. Cada módulo es una parte del

sistema, que se instala y funciona por separado, entrelazándose con otros módulos con los que intercambia información.

Objeto: Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad. Corresponden a los objetos reales del mundo que nos rodea, o a objetos internos del sistema.

Proceso: Conjunto de actividades que guían los esfuerzos de las personas implicadas para el cumplimiento de un objetivo.

Software: Se refiere a los programas y datos almacenados en un ordenador.

- Los programas dan instrucciones para realizar tareas al hardware o sirven de conexión con otro software.

Anexos

Anexo 1: Diagramas de clases

Diagrama de clase del caso de uso Modificar Proyecto

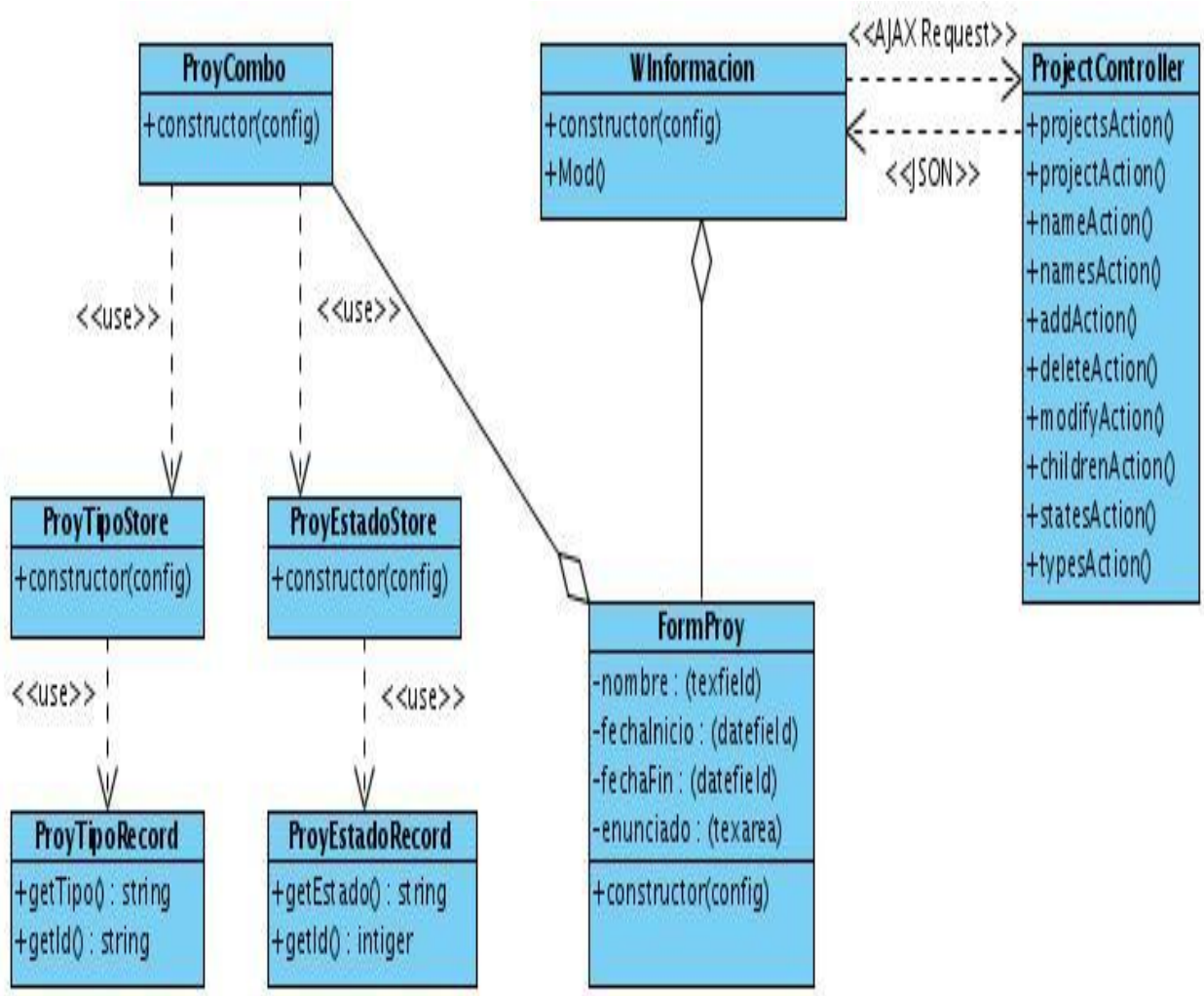


Figura 30 Diagrama de clase del CU_Modificar Proyecto.

Diagrama de clases del Caso de Uso Información de la Actividad

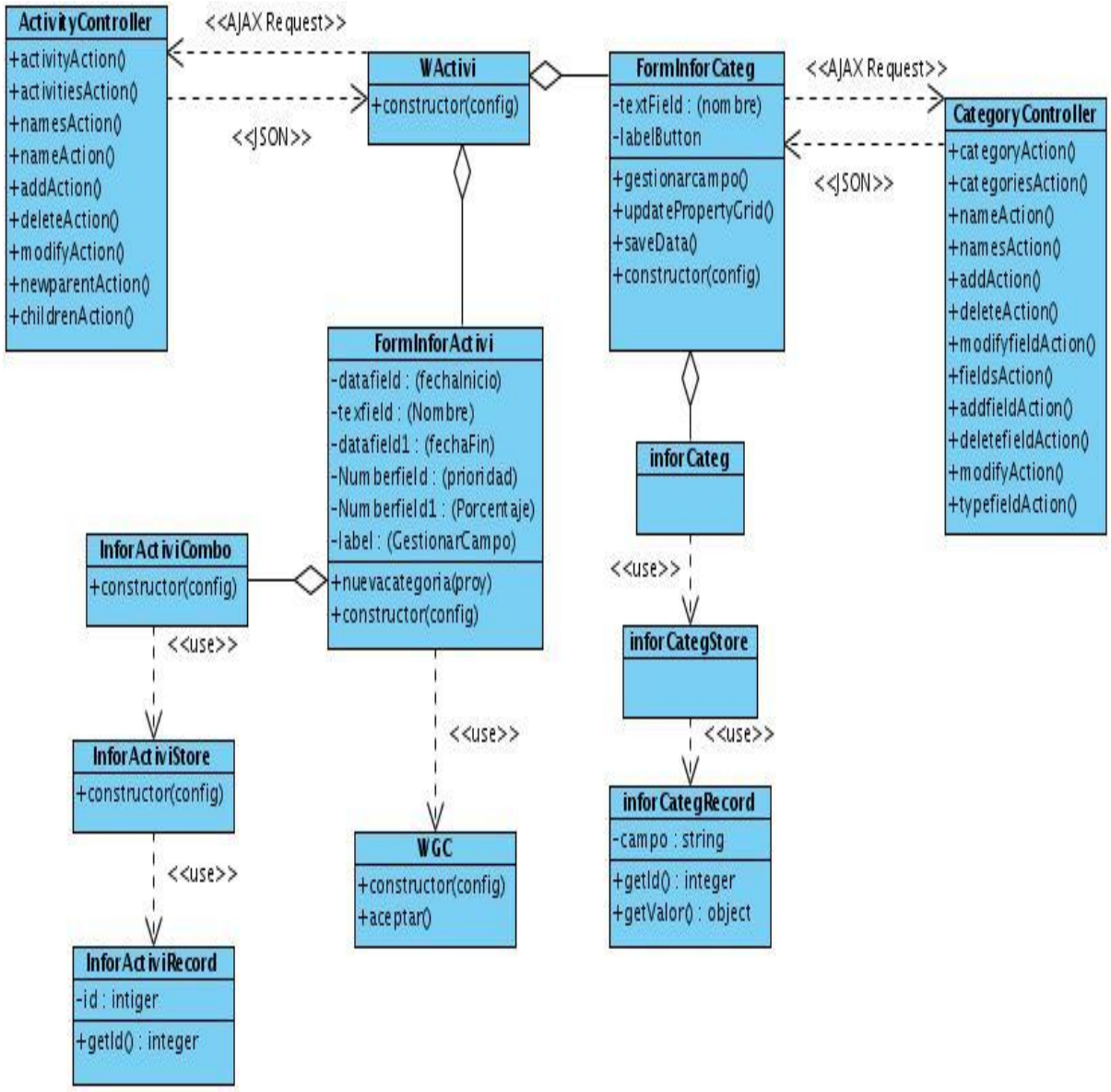


Figura 31: Diagrama de clase del CU_Información de la Actividad.

Diagrama de clases del Caso de Uso Modificar Actividad

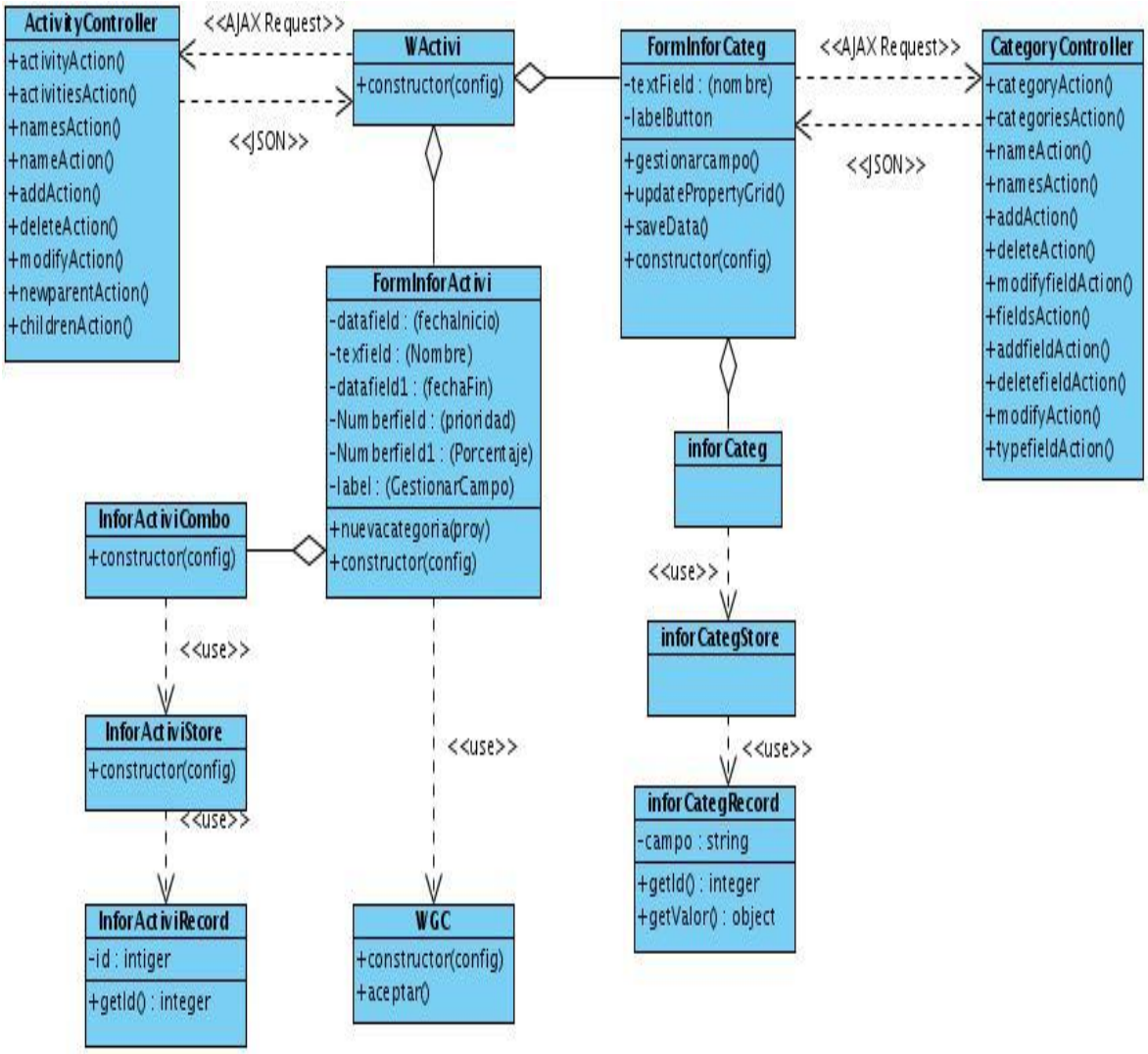


Figura 32: Diagrama de clases del CU_Modificar Actividad.

Anexo 2: Diagramas de secuencias

Diagrama de secuencia Caso de Uso Información de la actividad

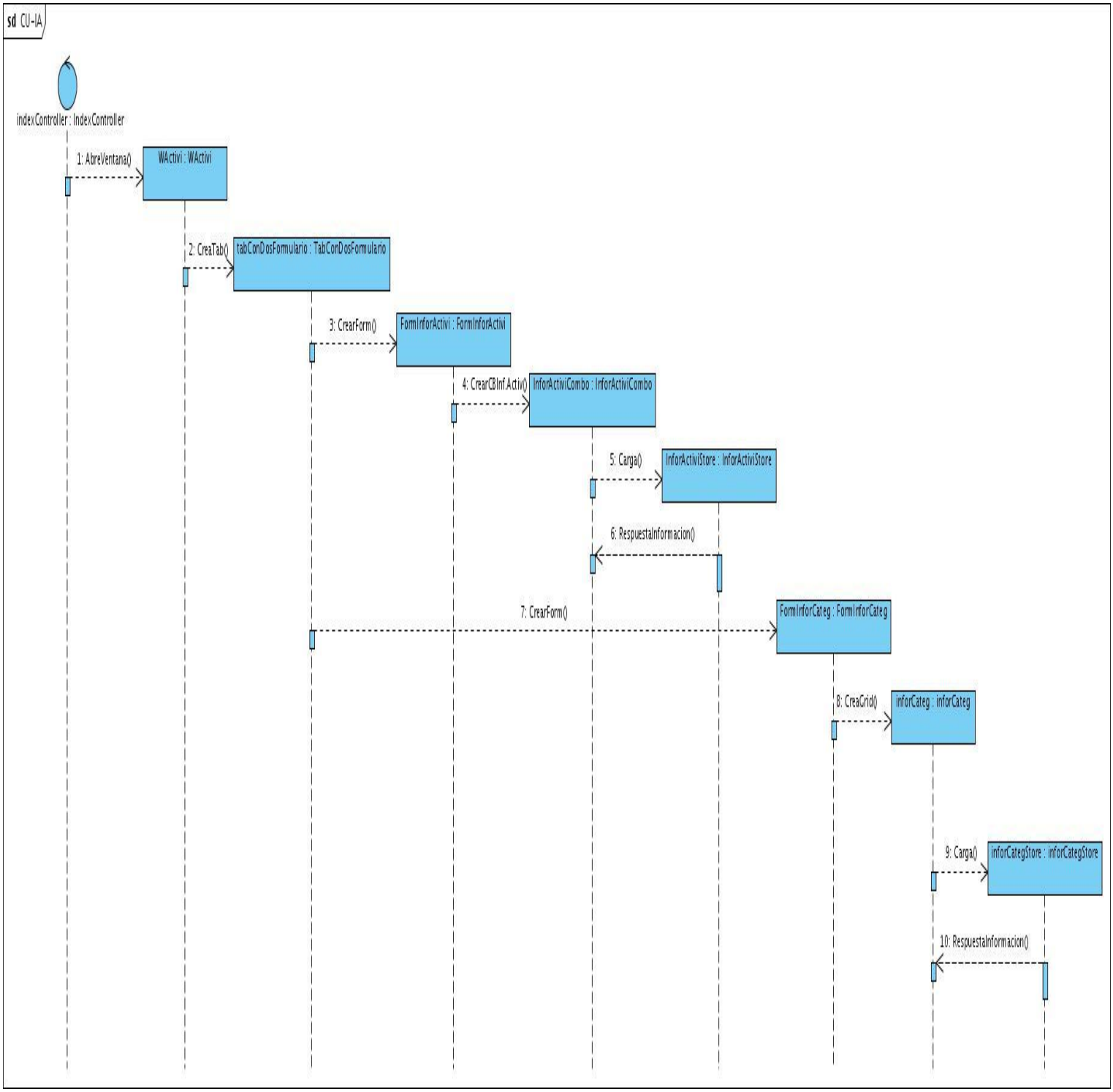


Figura 33: Diagrama de secuencia CU_Información de la actividad.