Universidad de las Ciencias Informáticas

“Facultad 6”

“Generador de rutinas para el Sistema Automatizado para la Gestión de la Información”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Miguel Ángel Cárdenas Hernández

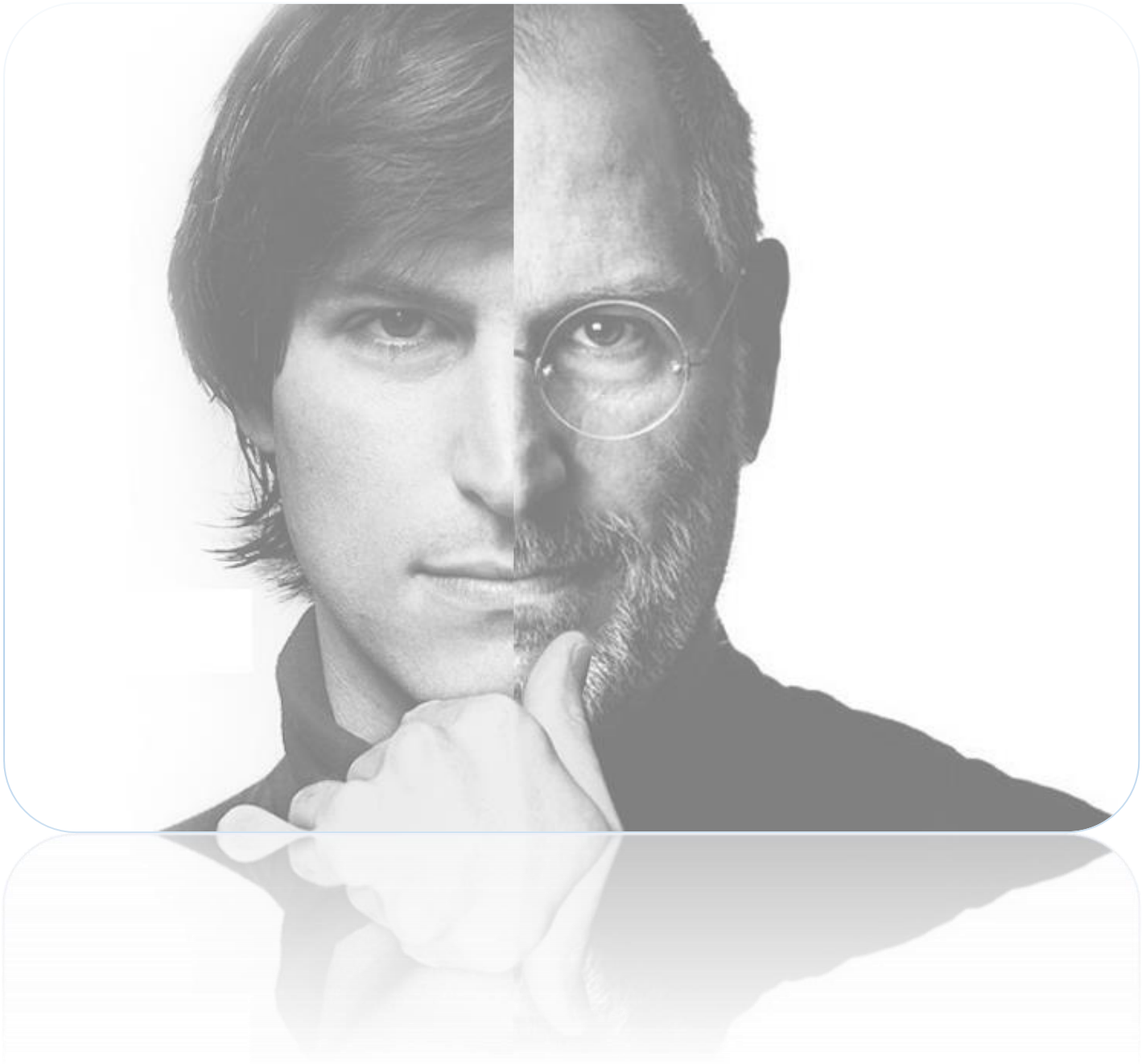
Lijandy Jiménez Armas

Tutores: Ing. Reynaldo Barceló Rodríguez

Ing. Dayana Joseph Smarth

La Habana, junio 2016

“Año 58 de la Revolución”



” El único modo de hacer un gran trabajo es amar lo que haces”

Steve Jobs

Declaración de autor

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

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2016.

Miguel Ángel Cárdenas Hernández

_____
Firma de Autor

Lijandy Jiménez Armas

_____
Firma de Autor

Ing. Dayana Joseph Smarth

_____
Firma de Tutora

Ing. Reynaldo Barceló Rodríguez

_____
Firma de Tutor

Datos de Autor

Tutora: Ing. Dayana Joseph Smarth

Universidad de las Ciencias Informáticas, La Habana, Cuba

Especialidad de graduación: Ingeniería Informática

Correo Electrónico: djoseph@uci.cu

Tutor: Ing. Reynaldo Barceló Rodríguez

Universidad de las Ciencias Informáticas, La Habana, Cuba

Especialidad de graduación: Ingeniería en Ciencias Informáticas

Correo Electrónico: barcelo@uci.cu

Autor: Miguel Ángel Cárdenas Hernández

Universidad de las Ciencias Informáticas, La Habana, Cuba

Correo Electrónico: macardenas@estudiantes.uci.cu

Autor: Lijandy Jiménez Armas

Universidad de las Ciencias Informáticas, La Habana, Cuba

Correo Electrónico: ljimenez@estudiantes.uci.cu

Dedicatoria de Migue:

Le dedico este logro de mi vida a mis padres por permitirme la oportunidad de estudiar esta carrera, a mi mamá por su fuerza de voluntad y a mi papá por su sacrificio.

Dedicatoria de Lija:

Me gustaría dedicarles este pequeño espacio de mi trabajo de diploma a esas personitas que fueron un todo para mí; para ellos estas pequeñas líneas:

Le dedico este logro de mi vida a esa persona que me trajo al mundo para ser feliz, a mi mami Miley que tanto la amo. También quiero dedicarles este logro a mis abuelitas Mirtha y Sarah que son un todo para mí, a mi papá por ser mi ejemplo y mi guía, a mi tío Ale por ser esa personita que siempre me ha guiado, a mis abuelos Piro y Bito, a Bárbaro por ser una gran persona para mí y a toda mi familia, muchos besos para todos.

Agradecimientos de Migue:

Agradezco a todas aquellas personas que de una forma u otra hicieron posible el logro de la creación de mi tesis, entre ellos a mis padres mi hermana, mi tío José Miguel, mi tía Maricela, Lijandy y sobre todo a esa persona que me ha apoyado todo este tiempo Mónica, que más que mi novia es mi amiga. A todos muchas gracias por formar parte de mí y de ayudarme a ser el profesional que soy HOY

.

Agradecimientos de Lija:

Desde muy pequeño soñé con ser un profesional, hoy este sueño se cumple. Muchos años han pasado para poder construirlo, siempre han estado presentes el esfuerzo, la dedicación, el sacrificio, como también he encontrado personas que han contribuido a que este sueño que tanto he esperado se hiciera realidad. A todas estas personas quisiera agradecerles:

A mi mamá Miley por ser una excelente madre y amiga por estar en los momentos buenos y en los momentos malos, a mis abuelitas Mirtha y Sarah por ser las mejores de este mundo, por darme la papa desde pequeño, por cuidarme y mimarme, y por educarme. A mi tía Milagro y Normita por apoyarme en el día a día, a mi papá Lázaro por ser una persona muy especial para mí, a mis abuelos Piro y Bito por querer tanto a ese nietecito que cuando por primera vez salía para el círculo le decían que parecía un hombrecito, muchos besos a los dos que a pesar de estar muy lejos de mí los voy a seguir queriendo con la vida. A mis primas Yalayna, Mayelin, Milaidy, Mileisy, Lenna, Lanny por ser parte de mí. A mis tíos Alberto, Ramón, Felo, Bello, Pablo, Tuto, Ilario por sus consejos de familia y de amigos que siempre me daban: mucho de ellos ya no están físicamente a mi lado, pero si quiero decirles donde quiera que estén que ya tienen al Lijandy que siempre quisieron. A mi tío Ale por ser ese tío que todo sobrino quisiera tener amigo y hermano te quiero tío, a Bárbaro por ser esa persona que desde que me conoció me dio sus consejos para seguir estudiando y alcanzar lo que siempre he querido. A mis primos Carlos, Orel, Robertico, Pablito a pesar de que no pude compartir con ellos cuando pequeños porque los veía como mis tíos por la edad que tenían cuando pequeño.

También me gustaría agradecer a esas personas que formaron parte de mi familia mediante el transcurso de mi vida, para mi amigo y hermano de la vida Dayan ese músico estelar que ha dado esta tierra, mi hermano muchas gracias por creer en mí, a mi amigo Daniel por darme sus consejos de su vida Universitaria y compartir momentos de fiesta y de

religión. A mi otra mamá Olga Lidia y a mi amigo Ramón por siempre estar al tanto del transcurso de las enseñanzas que he cursado, a su hija Danay, a mí amiga Isely a ambas por siempre alertarme de las cosas y confiar en nuestra amistad.

A mis tutores Dayana y Reynaldo por ser ejemplo y guía por demostrarme que siempre se puede lo que hay que ponerle empeño a las metas que nos trazamos y que nos enfrentamos, a mis otros tutores Dalilys y Carlos Chao por estar siempre presente en los momentos de buenos y malos, y por estar siempre al tanto del desarrollo de la tesis, por su apoyo incondicional.

Por supuesto no me gustaría nombras a esas personas que formaron parte de mi familia Universitaria a los chicos del 6103 Joyce, Orlando, Darian a mi compañero de mesa Eddy, Manuel, Yosvany Araño. También a la familia del 6304, 6404, 6504 y a los infiltrados del 6503 Leosdany, Leonardo, Eduardo, los Carlos, Victor, Yamil, Alejandro, Michel a las chicas del aula Indira y Mayi, a todos los quiero. A todas esas personas que de una forma u otra han formado parte de mi formación a la profe Yanelis, Telles, Yelenis, Alejandro al profe Jose. A mis compañeros de batalla en la FEU inmenso abrazo por formar parte de esas batallas del día a día. A los que de una forma u otra se han preocupado por el proceso de tesis, a aquellos que día a día me pregunta: "Cómo va esa tesis". A todos muchas gracias por formar parte de mí y de ayudarme a ser el profesional que soy HOY.

Resumen

La estadística adquiere relevancia en tanto hace posible transformar grandes cantidades de datos en información útil a los procesos administrativos como soporte a la toma de decisiones. En Cuba la Oficina Nacional de Estadística e Información (ONEI) realiza la gestión de los datos estadísticos a través del Sistema Automatizado para la Gestión de la Información (SAGI), desarrollado en convenio con la Universidad de las Ciencias Informáticas (UCI). Este sistema permite el diseño de formularios y encuestas para la captura de información y la generación de reportes mediante el Generador Dinámico de Reportes (GDR), lo que contribuye al proceso de toma de decisiones. Uno de los procesos fundamentales para la concepción de reportes es la generación de rutinas, algo que no es posible realizar hoy en SAGI. Este proceso se encuentra implementado en una herramienta externa, SIGETools, lo que obliga a los especialistas a utilizar dos o más aplicaciones para la obtención de reportes. El objetivo de la presente investigación es desarrollar un módulo para SAGI que permita la creación de rutinas. Para guiar dicha investigación se seleccionó como metodología de desarrollo de software OpenUP, UML 2.1 como lenguaje de modelado y Visual Paradigm 8.0 como herramienta CASE., los lenguajes de programación PHP 5 y JavaScript, Symfony 1.1.7 y ExtJS 3.4 como framework de desarrollo y para el tratamiento de las bases de datos PostgreSQL 9.3 y PgAdmin III. Además, se diseñaron e implementaron las clases del módulo aplicando buenas prácticas de los patrones de diseño. Se le realizaron pruebas al mismo, comprobando su correcto funcionamiento.

Palabras Claves

Datos estadísticos, generación de rutinas, reportes, Sistema Automatizado para la Gestión de la Información.

Abstract

The statistic becomes relevant as makes it possible to transform large amounts of data into useful administrative processes and support decision-making information. In Cuba the National Office of Statistics and Information (ONEI) performs the management of statistical data through the Automated System for Information Management (SAGI), developed in partnership with the University of Information Science (UCI). This system allows the design of forms and surveys to capture information and generate reports using the Dynamic Report Generator (GDR), contributing to the decision-making process. One of the fundamental processes for designing reports is generating routines, something that is not possible today with SAGI. This process is implemented in an external tool, SIGETools, forcing specialists to use two or more applications for obtain reports. The aim of this research is to develop a module for SAGI that allows the creation of routines. To guide this research was selected as software development methodology OpenUP, as UML 2.1 Modeling Language 8.0 and Visual Paradigm as CASE tool, Languages PHP 5 and JavaScript, ExtJS 3.4 and Symfony 1.1.7 programming and development framework. For the treatment of databases PostgreSQL 9.3 and PgAdmin III. In addition, they were designed and implemented classes applying good practices module design patterns. Tests were performed at the same, verifying proper operation.

Keywords

Statistical data, routines generation, reports, Automated System for Information Management.

Índice

Índice de Figuras

Índice de Tablas

Introducción

El análisis estadístico resulta fundamental para la recolección y agrupamiento de datos de diversos tipos para construir con ellos informes estadísticos que nos permitan conocer y analizar el comportamiento sobre diferentes temas, siempre desde un punto de vista cuantitativo. En Cuba dado el carácter esencialmente central de la administración del Estado, toda empresa, organismo, organización o institución que posea información de interés para la toma de decisiones de los órganos de la administración del Estado, está en la obligación de reportar información estadística bajo convenio con la Oficina Nacional de Estadística e Información (ONEI).

La ONEI ha priorizado la introducción de técnicas de almacenamiento y procesamiento de datos. En este contexto se inició de conjunto con la Universidad de las Ciencias Informáticas (UCI) el desarrollo de una aplicación que de soporte al marco metodológico del trabajo estadístico. Surgiendo el proyecto de informatización Sistema Automatizado para la Gestión de la Información (SAGI), este tiene como finalidad la gestión estadística en Cuba. Como parte de esta gestión comprende los procesos de diseño de plantillas de encuestas y formularios, para la captura de datos estadísticos, así como también permite organizar y exhibir los datos mediante reportes. Estos permiten dar un formato determinado a la información captada para mostrarla por medio de un diseño atractivo y que sea fácil de interpretar por los usuarios.

SAGI genera los reportes mediante el Generador Dinámico de Reportes (GDR) en su versión 1.8, para la obtención de los mismos el especialista debe crear un modelo (abstracción de la base de datos que contiene tablas, vistas y funciones). También debe implementar las consultas necesarias para obtener los datos que se desean visualizar a través del reporte. Posteriormente se selecciona un modelo y se procede a actualizar los metadatos de este, de manera que se puedan incluir las consultas implementadas o modificadas. En ocasiones los reportes difieren en algún parámetro, dígase provincia, mes, año, indicador y nombre de la empresa, aun cuando la diferencia entre los reportes sea mínima es necesario la repetición de todo el proceso para cada uno de estos, ya que la ONEI procesa anualmente un gran cúmulo de datos lo que trae como consecuencia que se emplee mucho tiempo y personal dedicado a ello.

Actualmente la ONEI para dar solución a esta situación emplea SIGETools, una aplicación que facilita la generación de funciones parametrizadas o rutinas, las cuales son funciones SQL que permiten parámetros de diferentes tipos, posibilitando la creación de reportes y a su vez estas puedan ser reutilizadas por los especialistas de la entidad. Aun cuando SIGETools ha agilizado el trabajo con los reportes hasta el momento, esta presenta varios inconvenientes:

- Está desarrollada con un lenguaje de licencia privativa, lo cual no cumple con las políticas de soberanía tecnológica en Cuba.

- Es una herramienta externa al SAGI la cual hace el trabajo más engorroso ya que se utilizan dos herramientas totalmente diferentes con el mismo fin.
- Es una herramienta de escritorio, por lo que debe ser instalada en todos los ordenadores que se utiliza el sistema.
- No es compatible con todas las distribuciones de Microsoft Windows y ninguna de Linux.
- No existe documentación sobre la herramienta, lo que dificulta el trabajo con la misma.
- No cuenta con un ciclo de soporte.

El empleo de dos herramientas totalmente diferentes tecnológicamente trae consigo la necesidad de una cantidad más elevada de especialistas con mayor tiempo disponible y mayor capacitación.

Debido a la necesidad de dar solución a la situación planteada se identifica como **problema de la investigación** que el Sistema Automatizado para la Gestión de la Información(SAGI) requiere para la generación de funciones parametrizadas el uso de una o más herramientas externas.

Donde el **objeto de estudio** está dirigido a la generación de rutinas, enmarcado en el **campo de acción** generación de rutinas en el Sistema Automatizado para la Gestión de la Información.

Para dar solución al problema de investigación planteado se define como **objetivo general** desarrollar el módulo generador de rutinas en el Sistema Automatizado para la Gestión de la Información para el proceso de creación de reportes en la Oficina Nacional de Estadística e Información.

Para darle cumplimiento al objetivo general se definen las siguientes **tareas de la investigación**:

- Análisis de los conceptos técnicos implicados en el desarrollo de la generación de rutinas y su utilidad para elaborar el marco teórico de la investigación.
- Análisis de la aplicación SIGETools para obtener información sobre el funcionamiento y el alcance de las mismas, mediante entrevistas con el cliente.
- Especificación de los requisitos funcionales y no funcionales del módulo generador de rutinas para el SAGI para describir las características del sistema, mediante entrevistas con los especialistas de la ONEI.
- Definición de la arquitectura del módulo generador de rutinas teniendo en cuenta la arquitectura del SAGI y aplicando patrones arquitectónicos.
- Elaboración del modelo de diseño para estructurar el módulo generador de rutinas, mediante la realización de diagramas de clases del diseño y diagramas de secuencia del diseño.
- Descripción del diagrama de componentes del módulo generador de rutinas para describir los elementos físicos del sistema y sus relaciones.

- Implementación de las funcionalidades del módulo generador de rutinas para dar respuesta a los requisitos del mismo, haciendo uso de los lenguajes de programación.
- Elaboración de los casos de prueba que serán aplicados al módulo generador de rutinas haciendo uso de técnicas y métodos de pruebas en los distintos niveles existentes.
- Ejecución de las pruebas a las funcionalidades del módulo generador de rutinas para detectar posibles no conformidades, mediante la ejecución de las pruebas de integración, funcionalidad y aceptación.

Preguntas de investigación:

- ¿Cuáles son los fundamentos teóricos en los que se basa el proceso de desarrollo del módulo generador de rutinas para el SAGI?
- ¿Cuáles son las características y capacidades que debe tener el módulo para que con este se permita una correcta integración al SAGI?
- ¿Cómo se debe estructurar el proceso de desarrollo del módulo generador de rutinas para SAGI que permita la gestión de los procesos que realiza la herramienta SIGETools para la creación de rutinas?
- ¿Cómo validar el correcto funcionamiento del módulo generador de rutinas del SAGI?

Para el cumplimiento del objetivo general planteado se utilizarán los siguientes **métodos de investigación**:

Métodos teóricos:

1. **Analítico-Sintético:** Es utilizado para descomponer los diversos procesos que se utilizan en la creación de una rutina en elementos por separado y profundizar en el estudio de cada uno de ellos de forma independiente, para luego sintetizarlos en la propuesta de solución.
2. **Inducción y deducción:** Es utilizado ya que la inducción es un procedimiento que permite a partir de hechos aislados, en este caso la generación de rutinas, arribar a proposiciones generales. La deducción es un procedimiento que permite a partir del estudio de conocimientos generales inferir casos particulares por un razonamiento lógico.

Métodos empíricos:

1. **Análisis Estático:** Permitió observar la estructura de SAGI, GDR y SIGETools, con el objetivo de clasificar las dificultades existentes y las funcionalidades que debe tener el sistema a implementar.
2. **Entrevista estructurada:** Se realiza a especialistas de la ONEI, con el objetivo de obtener información actualizada de los procesos que están informatizados y de las nuevas funcionalidades que se desean incorporar y mejorar; así como para definir los requisitos funcionales y no funcionales del mismo. Se utilizó la entrevista no estructurada debido a que no se basa en un cuestionamiento

rígido y es aplicado a personas que son especialistas en el tema abordado, utilizando la técnica de guía de entrevista (Ver anexo 1).

El Trabajo de Diploma está estructurado de la siguiente manera: Introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y bibliografía.

Capítulo 1: Fundamentación Teórica del módulo Generador de Rutinas para el SAGI: En este capítulo se presenta la definición del marco teórico de la investigación. En el mismo se abordan conceptos de rutinas y modelos. Se exponen características de sistemas informáticos existentes vinculados a la gestión de reportes. Se aborda el estudio de la metodología, herramientas, tecnologías y lenguajes a utilizar en el desarrollo del módulo.

Capítulo 2: Análisis y Diseño del módulo Generador de Rutinas para el SAGI: En el capítulo se describe la propuesta de solución para el problema de la investigación. Mediante el modelo de dominio se describe como se ve reflejado el negocio actual. Describiendo los principales conceptos del entorno que serán objeto de análisis para la realización de la fase de análisis y diseño del módulo para la generación de rutinas del SAGI. Posteriormente se hace el levantamiento de requisitos funcionales y no funcionales para darle cumplimiento al objetivo general de la investigación. Además, se muestran los diagramas de clases del diseño, por medio de los cuales se representa la estructura estática del sistema y los patrones de arquitectura y diseño utilizados.

Capítulo 3*:* Implementación y Pruebas del módulo Generador de Rutinas para el SAGI: En el capítulo se aborda sobre las actividades que se llevan a cabo durante la fase de implementación y pruebas. Se analizará el modelo de implementación, así como una descripción de cómo los elementos del modelo de diseño se implementan en términos de componentes. Además, se diseña y aplican las pruebas para comprobar el correcto funcionamiento del módulo.

Capítulo 1: Fundamentación Teórica del Generador de Rutinas para SAGI.

En este capítulo se presenta la definición del marco teórico de la investigación. En el mismo se abordan conceptos de rutinas y modelos. Se exponen características de sistemas informáticos existentes vinculados a la gestión de reportes. Se aborda el estudio de la metodología, herramientas, tecnologías y lenguajes a utilizar en el desarrollo del módulo.

1.1. Conceptos relacionados al dominio del problema.

Función: Una vez que se tiene una base de datos con información útil es importante saber cómo extraerla y hacer uso de ella. Uno de los mecanismos que se utiliza para esto son las funciones. Las mismas son objetos de la base de datos que están programados con un lenguaje procedural específico y que pueden facilitar la administración de la base de datos y la visualización de la información contenida en ella. Se compilan en el sistema gestor de base de datos cuando se crean, por tanto, se ejecutan con mayor rapidez que las instrucciones SQL individuales. En la implementación de una función se puede hacer uso de consultas y vistas, dos mecanismos de obtención de información que además pueden utilizarse individualmente (ibiblio, 2014).

Consulta: Las consultas tienen como objetivo recuperar datos específicos de las tablas, los cuales suelen estar dispersos y a través de ellas, se pueden ver en una sola hoja de datos. Este mecanismo de obtención de información permite agregar criterios para filtrar los datos hasta obtener solo los registros que se deseen. Las consultas pueden ser de lectura o escritura. Las de lectura (SELECT), simplemente recuperan los datos y hacen que estén disponibles para su uso. Las de escritura (INSERT, UPDATE, DELETE), pueden servir para crear tablas nuevas, agregar datos a tablas existentes y actualizar o eliminar datos (ibiblio, 2014).

Vista: La vista es una tabla virtual resultante de una consulta SQL en la cual se cargan los datos en el momento de ser llamada, sólo se almacena de ella la definición. A través de su implementación se pueden obtener datos de una o varias tablas. Presenta desventajas en cuanto a rendimiento, porque el Sistema Gestor de Bases de Datos (SGBD) debe traducir las consultas realizadas a la vista en consultas realizadas a las tablas fuentes, por lo que pueden tardar mucho tiempo en completarse (ibiblio, 2014).

Reporte: Un reporte es un informe o una noticia. Este tipo de documento (que puede ser impreso, digital o audiovisual) pretende transmitir una información, aunque puede tener diversos objetivos. Generalmente agrupan información de acuerdo a un interés específico, muestran el resultado de una búsqueda determinada. Permiten dar un formato determinado a los datos para mostrarlos por medio de un diseño atractivo y que sea fácil de interpretar por los usuarios. De esta forma, confiere una mayor utilidad a los datos haciendo posible sintetizar la información esencial (Anglada, 2013).

Modelo: Un modelo de datos es un sistema formal y abstracto que permite describir los datos de acuerdo con reglas y convenios predefinidos o podríamos decir que es un conjunto de concepto que permiten describir a distintos niveles de abstracción la estructura de una base de datos (Marin, 2008).

Rutina: Una rutina también conocida como función o subrutina, entre otros nombres es una secuencia invariable de instrucciones que forma parte de un programa y que puede utilizarse una y otra vez. En este sentido, la rutina se presenta como un sub algoritmo dentro del algoritmo principal (el programa), en general, se utilizan rutinas para mejorar el rendimiento global del sistema de gestión de bases de datos, permitiendo que las funciones de las aplicaciones se realicen en el servidor de bases de datos. La proporción de mejora obtenida con estos esfuerzos está limitada, en cierta medida, por el lenguaje elegido para escribir una rutina (IBM, 2009). A continuación, se muestra un ejemplo:

```
CREATE FUNCTION fn_TopVentasRango (@iFechaInicial as int, @iFechaFinal as int) RETURNS TABLE
AS RETURN (SELECT CodigoArticulo, Row_number () OVER (ORDER BY Sum (unitats) DESC) AS nRanquing
 FROM Ventas WHERE FechaVenta BETWEEN @iFechaInicial AND @iFechaFinal)
```

1.2. Sistemas informáticos existentes que sustentan la investigación

Sistema Automatizado para la Gestión de la Información (SAGI):

SAGI fue desarrollado de conjunto con la ONEI de forma tal que contempla el marco metodológico y operativo de esta. Como sistema está integrado por módulos altamente especializados que permiten la gestión y la captación de la información estadística. Este sistema está compuesto por los siguientes módulos: Gestión de Configuración, Gestor de Plantillas, Entrada de Datos, Administración, Herramientas, Seguridad y Generador de Reportes, este último componente se encuentra embebido en el sistema y permite la visualización de la información almacenada. Actualmente SAGI se encuentra en explotación en la ONEI, así como en todas sus entidades provinciales y municipales, el cual ha tenido una gran aceptación en este centro.

Generador Dinámico de Reportes 1.8 (GDR):

El Sistema Generador Dinámico de Reportes 1.8 (GDR) es una herramienta desarrollada por el centro DATEC para la realización de los reportes ejecutivos. En el contexto de SAGI es la herramienta para la consulta de la información estadística. El sistema cubre el ciclo de vida de un reporte desde su creación, hasta su consulta y distribución. Entre las principales funcionalidades están: la gestión de los modelos de datos para el reporte (fuente de datos), el diseño visual de consultas, la visualización de los reportes, la gestión de los reportes y su distribución por correo electrónico, la gestión intensiva de la seguridad.

La solución de reportes de DATEC agrega valor al SAGI a partir de su constante evolución y mejora con nuevas prestaciones como la generación de distintos tipos de gráficos, la incorporación de funciones

estadísticas para consultar la información y de tipos especializados de reportes. Una cuestión adicional de este producto es su capacidad de integración con sistemas de terceros a través de una interfaz de programación de aplicaciones (API, por las siglas en inglés de Application Programing Interface) esto facilita que la consulta de reportes previamente elaborados pueda realizarse embebido en otro sistema web de una manera fácil.

SIGETools

Aplicación de escritorio desarrollada por especialistas de la ONEI que permite consultar los gestores de bases de datos y generar rutinas con la información que estos manejan. Cuenta con 7 módulos Cortes, Niveles, Modelos, Modelos Virtuales, Usuario y Configuración en estos se puede realizar la administración de cortes, definición de niveles, visualizar los modelos; reales y virtuales. La creación de estos modelos permite una generación de reportes de forma rápida y con una amplia gama de opciones que facilitan el diseño de reportes por los usuarios no informáticos. Esta herramienta da solución al problema de obtener diferentes reportes en los sistemas de gestión de la información. Es un producto capaz de mostrar reportes, con el objetivo de ayudar a la tomar decisiones, realizar estudios investigativos, o sencillamente consultar información propia del negocio para el cual fue concebido. Es una aplicación que permite a sus clientes consultar los gestores de bases de datos de sus organizaciones y poder generar rutinas. Actualmente SIGETools se encuentra en explotación en la ONEI. Al ser una aplicación de escritorio, no se puede vincular al SAGI trayendo consigo la utilización de dos aplicaciones para lograr un resultado para la empresa. Esto conlleva a que se emplee más tiempo en el diseño y realización de las rutinas los cuales son imprescindibles a la hora de facilitar la creación de los reportes, dichos reportes que son fundamentales en la toma de decisiones de la institución.

Por tanto, se propone incluir en SAGI un módulo Generador de Rutinas que logre realizar las principales funcionalidades del SIGETools, dígase la gestión de modelos virtuales, niveles, cortes y otras definidas por el cliente permitiendo así realizar todo el proceso de creación de funciones parametrizadas (rutinas) dentro del mismo y de esta manera inhibir el uso de herramientas externas facilitando el proceso de creación de rutinas y reportes en la ONEI.

Con la creación del módulo se organiza y centraliza la generación de rutinas, facilitando la labor de los especialistas de la ONEI al estar más familiarizados en el trabajo con SAGI. Además, se mejora el proceso de peticiones a la base de datos ya que no serían dos o más herramientas gestionando información sino solo una. También posibilita no tener que desplegar una nueva herramienta en las demás extensiones de la ONEI teniendo todo el proceso de creación de reportes en una misma aplicación.

1.3. Metodología, tecnologías y herramientas a emplear en la solución

1.3.1. Metodología de desarrollo de software

La metodología para el desarrollo de software en un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Una metodología para el desarrollo de software comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado (INTECO, 2010).

Una definición estándar de metodología puede ser el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. Determina los pasos a seguir y cómo realizarlos para finalizar una tarea (INTECO, 2010), dentro de estas metodologías se encuentra OpenUP.

OpenUP: Es un proceso modelo y extensible, dirigido a gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo (eclipse, 2011).

Se escoge esta metodología ya que:

- Es apropiado para proyectos pequeños y de bajos recursos permite disminuir las probabilidades de fracaso en los proyectos e incrementar las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

1.3.2. Lenguaje de Modelado

Lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados (disca, 2012).

Es importante remarcar que UML es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir.

UML 2.1: UML 2 avanza la especificación UML éxito, y se está convirtiendo rápidamente en el estándar aceptado para especificar, documentación y visualización de sistemas de software. También se utiliza para el modelado de sistemas no de software, y se implementa ampliamente en la mayoría de los sectores de la industria, incluyendo las finanzas, militar y de ingeniería. (uml-diagrams, 2010)

Se utiliza este lenguaje de modelado por proporcionar orientación en cuanto al orden de las actividades de un equipo, dirigir las tareas de los desarrolladores individuales y del equipo en su conjunto, por ofrecer criterios para el seguimiento, la medición de los productos y las actividades de un proyecto.

1.3.3. Herramientas CASE

Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas son de gran ayuda en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras *(MARTÍNEZ, 2016)*

Visual Paradigm 8.0

Visual Paradigm (anteriormente VP-UML) es una herramienta UML. La herramienta está diseñada para una amplia gama de usuarios, incluyendo ingenieros de software, analistas de sistemas, analistas de negocios y arquitectos de sistemas, o para cualquier persona que esté interesada en la construcción de forma fiable los sistemas de software a gran escala con un enfoque orientado a objetos (Paradigm, 2010).

Se escoge la herramienta Visual Paradigm por soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas, sirviendo como guía y facilitando el proceso de desarrollo de todas las fases de creación del módulo (STEWART, 2016).

1.3.4. Lenguaje de Programación

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos de hardware y software existentes *(KANEIWA, y otros, 2015)*.

PHP5: Acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje de 'scripting' de propósito general y de código abierto que está especialmente pensado para el desarrollo web y que puede ser embebido en páginas HTML. Su sintaxis recurre a C, Java y Perl, siendo así sencillo de aprender. El objetivo principal de este lenguaje es permitir a los desarrolladores web escribir dinámica y rápidamente páginas web generadas; aunque se puede hacer mucho más con PHP *(KANEIWA, y otros, 2015)*.

JavaScript: JavaScript es un lenguaje de scripting multiplataforma y orientado a objetos. Es un lenguaje pequeño y liviano. Dentro de un ambiente de host, JavaScript puede conectarse a los objetos de su ambiente y proporcionar control programático sobre ellos.

JavaScript contiene una librería estándar de objetos, tales como Array, Date, y Math, y un conjunto central de elementos del lenguaje, tales como operadores, estructuras de control, y sentencias. El núcleo de JavaScript puede extenderse para varios propósitos, complementándolo con objetos adicionales (MDN, 2009).

1.3.5. Marco de trabajo

Un marco de trabajo o framework es un diseño abstracto orientado a objetos para un determinado tipo de aplicación, es un patrón arquitectónico que proporciona una plantilla extensible para un tipo específico de aplicaciones. Es un conjunto cohesivo de interfaces y clases que colaboran para proporcionar los servicios de la parte central e invariable de un subsistema lógico. El marco de trabajo diseñado posee una arquitectura orientada a componentes y está compuesto por cinco paquetes principales: Núcleo, Servicios, Persistencia, Componentes y Aplicación de Usuario. Cada paquete está compuesto por componentes y estos a su vez definen una interfaz de comportamiento con el objetivo de extender sus funcionalidades y adaptarse a posibles escenarios (Anglada, 2013)**.**

Symfony 1.1.7: Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web, este separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (Potencier, 2009).

- La capa de internacionalización que incluye Symfony permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos.
- La capa de presentación utiliza plantillas y layouts que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del framework. Los helpers incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- Los formularios incluyen validación automatizada y relleno automático de datos "repopulation", lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.
- Los datos incluyen mecanismos de escape que permiten una mejor protección contra los ataques producidos por datos corruptos.
- La gestión de la caché reduce el ancho de banda utilizado y la carga del servidor.

- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- El sistema de enrutamiento y las URL limpias permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.
- El soporte de e-mail incluido y la gestión de APIs permiten a las aplicaciones web interactuar más allá de los navegadores.

ExtJS 3.4: De acuerdo a la definición, ExtJS es una librería JavaScript que permite construir aplicaciones complejas en Internet. Esta librería incluye:

- Componentes UI del alto desempeño y personalizables.
- Modelo de componentes extensibles.
- Un API fácil de usar.
- Licencias open source y comerciales.

Una de las grandes ventajas de utilizar ExtJS es que permite crear aplicaciones complejas utilizando componentes predefinidos, así como un manejador de layouts similar al que provee Java Swing, gracias a esto provee una experiencia consistente sobre cualquier navegador, evitando el tedioso problema de validar que el código escrito funcione bien en cada uno (Firefox, IE, Safari, etc.) (CHEN, 2011).

Como marcos de trabajo se utilizarán Symfony 1.1.7 y ExtJS 3.4 teniendo en cuenta que la programación del lado del servidor en SAGI se llevó a cabo con Symfony 1.1.7 y que ExtJS 3.4, en este caso utilizado para la programación de la interfaz, están diseñados basados en el patrón arquitectónico Modelo-Vista-Controlador, que facilita la separación de estos elementos posibilitando un balance entre Cliente – Servidor, de esta forma la carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo. También contienen una base de componentes amplia que permite la reutilización y extensión de los mismos, así como una extensa documentación.

1.3.6. Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado(IDE), es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI) (IBM, 2009)***.***

NetBeans 8.0.2*:* Es un entorno de desarrollo gratuito y de código abierto que en el momento de escribir este artículo está en su versión 8.0.2. Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web, o para dispositivos móviles. Da soporte a las siguientes tecnologías, entre otras: Java, PHP, Groovy, C/C++, HTML5. Además, puede instalarse en varios sistemas operativos: Windows, Linux, Mac OS, etc. (genbetadev, 2014).

Este IDE de desarrollo se usa fundamentalmente por ser multilenguaje, pero además por ser de código abierto y gratuito. Por lo cual es la herramienta idónea para el desarrollo del módulo.

1.3.7. Sistema Gestor de Bases de Datos (SGBD)

PostgreSQL 9.3: PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código (OBE, y otros, 2015). Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (OBE, y otros, 2015).

PgAdmin III: es una herramienta de código abierto para la administración de bases de datos PostgreSQL y derivados (EnterpriseDB, PostgresPlus, Advanced Server y GreenplumDatabase. Responde a las necesidades de la mayoría de los usuarios, desde escribir simples consultas SQL hasta desarrollar bases de datos complejas. La interfaz gráfica soporta todas las características de PostgreSQL y hace simple la administración. Está disponible en más de una docena de lenguajes y para varios sistemas operativos, incluyendo Microsoft Windows, Linux, FreeBSD, Mac OSX y Solaris. Soporta versiones de servidores 7.3 y superiores. Versiones anteriores a 7.3 deben usar PgAdmin II (MICHAUD, 2015).

1.3.8. Servidor web

Apache 2.2: Un servidor web es, como su nombre indica, un software instalado en el equipo con todas las condiciones necesarias para servir o entregar páginas web que le sean solicitadas por un navegador, asegurando que se muestren y representen todos los elementos necesarios para su correcto funcionamiento y visualización. Existen varios tipos de servidores web, Apache es un software de código abierto, libre de uso y totalmente configurable, es en este momento el más utilizado en la red, ya sea en plataformas Linux o Windows (MORO, y otros, 2013).

El servidor está dirigido a servir a una gran cantidad de plataformas web que trabajan sobre sistemas operativos como Unix, Windows, Linux, Solaris, Novell NetWare, FreeBSD, Mac OS X, Microsoft Windows, OS / 2 (PAYER, y otros, 2015).

Conclusiones del Capítulo

En el capítulo que finaliza se establecieron las herramientas y tecnologías que posibilitaran un correcto desarrollo del módulo Generador de Rutinas para el SAGI seleccionándose como metodología de desarrollo de software OpenUP, la cual guiará todo el proceso de creación del sistema propuesto. Eligiendo UML 2.1 como lenguaje de modelado y Visual Paradigm 8.0 como herramienta CASE. El lenguaje de programación a utilizar, PHP 5 y JavaScript, para facilitar el desarrollo se utilizarán los framework Symfony 1.1.7 y ExtJS 3.4, NetBeans 8.0.2 como IDE. Para el tratamiento de las bases de datos PostgreSQL 9.3 y PgAdmin III, y de servidor de aplicaciones Apache 2.2. Cada una de estas herramientas y tecnologías además de ser utilizadas en la programación del SAGI, posee características que las hacen idóneas para el trabajo con ellas durante el proceso de desarrollo del mismo.

Capítulo 2: Análisis y Diseño del Generador de Rutinas para SAGI.

El diseño de un producto o sistema es fundamental para un desarrollo exitoso del mismo, en este se desarrollan, revisan y documentan los requisitos del producto y los detalles procedimentales de su implementación. En el capítulo se describe la propuesta de solución para el problema de la investigación. Mediante el modelo de dominio se describe la estructura que se desea representar en el módulo a desarrollar. Describiendo los principales conceptos del entorno que serán objeto de análisis para la realización de la fase de Análisis y Diseño del módulo para la Generación de rutinas del SAGI. Posteriormente se realiza el levantamiento de requisitos funcionales y no funcionales para darle cumplimiento al objetivo general de la investigación. Además, se muestran los diagramas de clases del diseño, por medio de los cuales se representa la estructura estática del sistema y los patrones de arquitectura y diseño utilizados en el sistema.

2.1. Modelo del dominio

El modelo de dominio puede ser tomado como el punto de partida para el diseño del sistema ya que permite utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema. El modelo de dominio es utilizado como un medio para comprender el sector de negocios al cual el sistema va a servir (Larman, 2008).

2.1.1. Diagrama conceptual del Modelo de Dominio

El presente modelo de dominio tiene como objetivo contribuir a la comprensión del módulo para la generación de rutinas para SAGI.

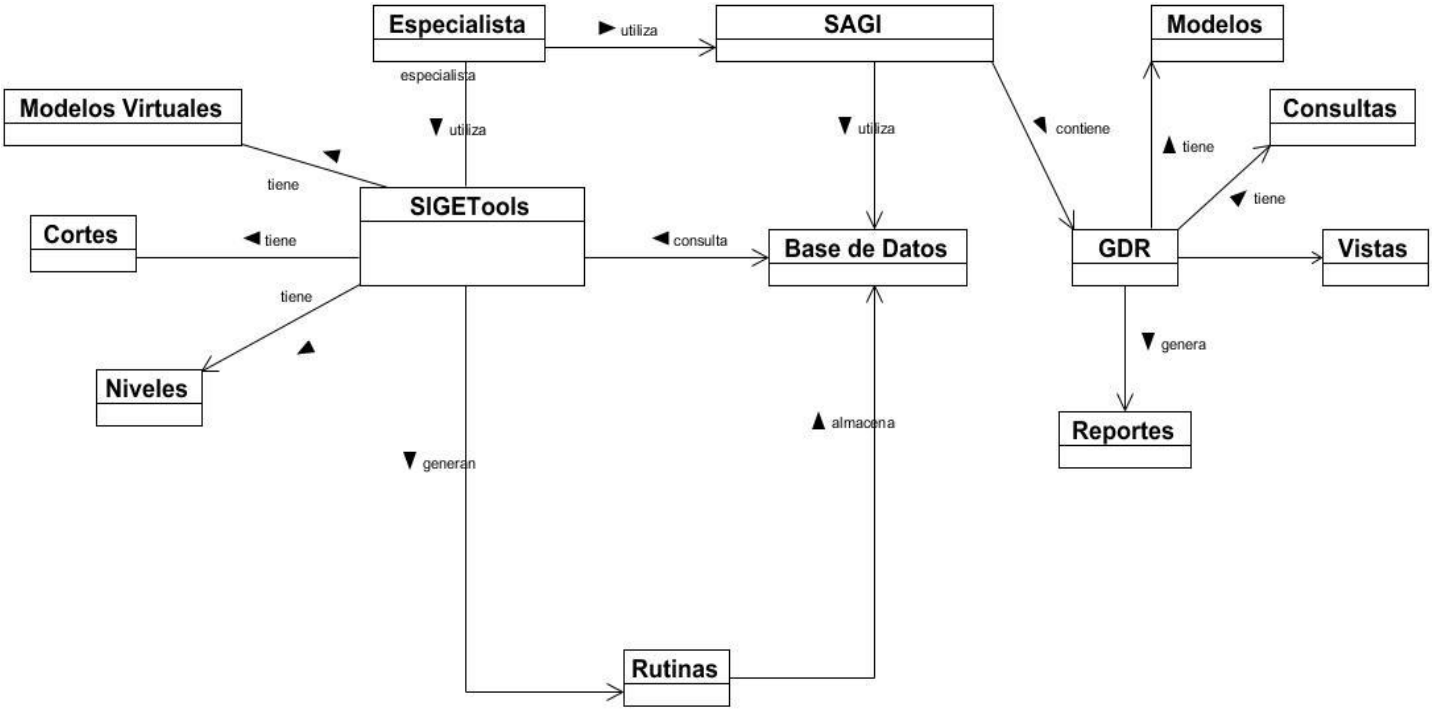


Fig. 1: Modelo Dominio

2.1.2. Definición de clases del modelo del dominio

Nivel: Se define como Nivel en la ONEI como una forma de subdividir las entidades para poder ejercer un control de los diferentes contornos, esto permite una mayor organización en el momento de obtener mayores y mejores resultados a diferentes niveles, dígase Provincia, Municipio y Unidad de Observación.

Corte: Se define como Corte en la ONEI aquella acción o actividad independiente y objetiva de supervisión y consultoría diseñada para agregar y mejorar las operaciones de una organización. Ayuda a la empresa a cumplir sus objetivos aportando un enfoque sistemático y disciplinado para evaluar y mejorar la eficiencia de los procesos de gestión de información, control y gobierno.

Modelo: Abstracción de la base de datos que contiene tablas, vistas y funciones.

Modelo Virtual: Nuevo modelo que se crea mediante la gestión de los modelos ya existentes.

Consulta: Las consultas pueden ser de lectura o escritura. Las de lectura (SELECT), simplemente recuperan los datos y hacen que estén disponibles para su uso. Las de escritura (INSERT, UPDATE, DELETE), pueden servir para crear tablas nuevas, agregar datos a tablas existentes y actualizar o eliminar datos.

Vista: La vista es una tabla virtual resultante de una consulta SQL en la cual se cargan los datos en el momento de ser llamada, sólo se almacena de ella la definición.

Rutina: Una rutina también conocida como función o subrutina, entre otros nombres es una secuencia invariable de instrucciones que forma parte de un programa y que puede utilizarse una y otra vez.

2.2. Especificación de los requisitos del sistema

Los requisitos para un sistema de software determinan lo que hará el sistema y definen las restricciones de su operación e implementación. Se pueden clasificar en: funcionales y no funcionales. Partiendo de la descripción de clases representadas en el Modelo de Dominio y las necesidades del cliente, fueron determinados los requisitos funcionales y no funcionales del sistema (JACOBSON, y otros, 2004).

2.2.1. Requisitos funcionales

Son capacidades o condiciones que el sistema debe cumplir, se obtienen las de las actividades que serán objeto de automatización. Estas actividades no son exactamente los requisitos funcionales, pero sí son el punto de partida para identificar qué debe hacer el sistema. Los requisitos funcionales se mantienen invariables sin importar con que propiedades o cualidades se relacionen (JACOBSON, y otros, 2004).

RF-1 Adicionar nivel.

Descripción: El sistema debe permitir adicionar un nivel mediante una interfaz en la cual el usuario insertará los datos. El sistema valida los datos entrados, y si todo es correcto crea el nivel, sino devuelve las excepciones en la respuesta.

Entrada: se especifican los detalles del nuevo nivel que se va a adicionar (Nombre, Orden, SubStr, Descripción, Acumulado, Tipo de Origen, Clasificador, Tipo de Dato, Es maestro y Acepta columna.).

RF-2 Listar nivel.

Descripción: El sistema debe permitir visualizar la respuesta de la petición realizada, mostrando una lista de los niveles existentes en el sistema.

Salida: listado de los niveles.

RF-3 Eliminar Nivel.

Descripción: El sistema verifica en la base de datos la existencia del nivel a eliminar enviado por parámetros, si la verificación es satisfactoria elimina el nivel de la base de datos del servidor, sino devuelve las excepciones en la respuesta.

Entrada: selección del nivel que se desea eliminar.

RF-4 Buscar Nivel

Descripción: El sistema busca en la base de datos del servidor el nivel especificado por el tipo enviado y lo devuelve, sino devuelve la excepción en la respuesta.

Salida: Muestra el nivel según los datos pasados por parámetros.

***RF-*5** Modificar Nivel.

Descripción: El sistema comprueba la existencia del nivel a modificar, si no existe muestra una excepción, de otra forma debe permitir editar un nivel mediante una interfaz en la cual el usuario insertara los datos. El sistema valida los datos entrados, y si todo es correcto modifica el nivel.

RF-6 Exportar Nivel

Descripción**:** El nivel es seleccionado por el usuario y luego es exportado al formato XML.

Entrada: selección del nivel que se desea exportar.

RF-7 Importar Nivel

Descripción: El usuario selecciona el o los niveles y luego son importados en formato XML.

Entrada: selección del archivo XML.

RF-8. Salvar corte

Descripción: El sistema debe permitir salvar (adicionar) un nuevo corte para una posible modificación. El sistema debe permitir salvar un corte mediante una interfaz en la cual el usuario insertará los datos. El sistema valida los datos entrados, y si todo es correcto crea el corte, sino devuelve las excepciones en la respuesta.

Entrada: se especifican los detalles del nuevo corte que se va a salvar (Nombre de corte y tipo de corte).

RF-9. Listar corte.

Descripción: El sistema debe permitir visualizar la respuesta de la petición realizada, mostrando una lista de los cortes existentes en el sistema.

Salida: listado de los cortes.

RF-10. Eliminar Corte.

Descripción: El sistema verifica en la base de datos la existencia del corte a eliminar enviado por parámetros, si la verificación es satisfactoria elimina el corte de la base de datos del servidor, sino devuelve las excepciones en la respuesta.

Entrada: Selección del corte que se desea eliminar.

RF-11. Buscar Cortes.

Descripción: El sistema busca en la base de datos del servidor el corte especificado por el parámetro enviado y lo devuelve, sino devuelve la excepción en la respuesta.

Entrada: Los criterios de búsqueda pueden ser: nombre, el id, la visibilidad o el tipo de parámetro.

Salida: listado de los cortes que cumplan con el parámetro especificado.

RF-12. Modificar Corte.

Descripción: El sistema debe permitir mediante una interfaz modificar el corte seleccionado, donde el usuario insertará los datos. El sistema valida los datos entrados por el usuario, y si todos son correctos se modifica el corte.

Entrada: Se modifican los detalles del corte seleccionado (Nombre de corte y tipo de corte).

RF-13 Listar Nivel Fila en Corte:

Descripción: El sistema debe permitir listar todos los niveles fila que se adicionan a la hora de generar un corte

Salida: listado de los niveles fila.

RF-14. Adicionar Niveles Fila en Corte.

Descripción: El usuario selecciona los niveles fila a incluir, si la validación es satisfactoria incluye el o los niveles, en caso de no serlo muestra una excepción.

Entrada: Se incluye el o los niveles seleccionados al corte

RF-15. Eliminar Nivel Fila en Corte.

Descripción: El usuario selecciona el nivel fila a excluir, si la validación es satisfactoria excluye el nivel, en caso de no serlo muestra una excepción.

Salida: Se elimina de la lista de niveles a incluir el nivel fila seleccionado.

RF-16 Exportar Corte

Descripción: El corte es seleccionado por el usuario y luego es exportado al formato XML.

Entrada: selección del corte que se desea exportar.

Salida: archivo XML.

RF-17 Importar Corte

Descripción**:** El corte es seleccionado por el usuario y luego es importado al formato XML.

Entrada: selección del archivo XML.

RF-18. Crear modelo virtual de tipo Submodelo.

Descripción: El sistema permite que el usuario seleccione un modelo mediante una interfaz en la cual insertará el código de fila, la descripción, el modelo a utilizar. El sistema valida los datos entrados, y si todo es correcto crea el submodelo, sino devuelve las excepciones en la respuesta.

Entrada: se especifican los detalles del nuevo modelo que se va a crear (definir modelo base y escoger modelos de los cuales se obtendrán los parámetros).

RF-19 Crear modelo virtual de tipo Unión.

Descripción: El sistema debe permitir adicionar una unión de modelos mediante una interfaz en la cual el usuario insertara el código de fila y la descripción de los modelos que necesita unir. El sistema valida los datos entrados, y si todo es correcto crea la unión de modelos, sino devuelve las excepciones en la respuesta.

Entrada: se especifican los detalles del nuevo modelo que se va a crear (definir modelo base y escoger modelos de los cuales se obtendrán los parámetros).

RF-20 Crear modelo Virtual de tipo Nuevo.

Descripción: El sistema debe permitir adicionar un modelo virtual de tipo nuevo, este permitirá que se cree un nuevo modelo a partir de parámetros de diferentes modelos existentes. Luego de añadido los parámetros el sistema valida los datos entrados, y si son correctos crea el modelo de tipo nuevo, sino devuelve una excepción en la respuesta.

Entrada: se especifican los detalles del nuevo modelo que se va a crear (definir modelo base y escoger modelos de los cuales se obtendrán los parámetros).

RF-21 Crear modelo virtual de tipo Intermodelo.

Descripción: El sistema permite que el usuario mediante una interfaz seleccione un modelo base y a partir de este incluyendo el código de fila puede incluir los aspectos de otros modelos existentes. El sistema valida los datos entrados, y si todo es correcto crea el intermodelo, sino devuelve las excepciones en la respuesta.

Entrada: se especifican los detalles del nuevo modelo que se va a crear (definir modelo base y escoger modelos de los cuales se obtendrán los parámetros).

RF-22 Eliminar Modelo Virtual.

Descripción: El sistema verifica en la base de datos la existencia del modelo virtual a eliminar enviado por parámetros, si la verificación es satisfactoria elimina el modelo virtual de la base de datos del servidor, sino devuelve las excepciones en la respuesta.

Entrada: selección del modelo que se desea eliminar.

RF-23 Visualizar código SQL en Intermodelo.

Descripción: El sistema permite que el usuario visualice el código SQL de la consulta que hace referencia al modelo virtual Intermodelo.

Salida: código SQL del modelo

RF-24 Visualizar código SQL en Unión de Modelos.

Descripción: El sistema permite que el usuario visualice el código SQL de la consulta que hace referencia al modelo virtual Unión de Modelos.

Salida: código SQL del modelo

RF-25 Visualizar código SQL en Submodelo.

Descripción: El sistema permite que el usuario visualice el código SQL de la consulta que hace referencia al modelo virtual Submodelo.

Salida: código SQL del modelo

RF-26 Visualizar código SQL en Modelo Genérico.

Descripción: El sistema permite que el usuario visualice el código SQL de la consulta que hace referencia al modelo virtual Genérico.

Salida: código SQL del modelo

RF-27 Incluir aspectos de Modelos Virtuales.

Descripción: El usuario selecciona los aspectos de los modelos base a incluir, si la validación es satisfactoria incluye los aspectos deseados, en caso de no serlo muestra una excepción.

Entrada: Se incluye el o los aspectos seleccionados.

RF-28 Eliminar aspectos de Modelos Virtuales.

Descripción: El usuario selecciona el aspecto a eliminar, si la validación es satisfactoria elimina el aspecto, en caso de no serlo muestra una excepción.

Salida: Se elimina de la lista de aspectos a incluir en el nuevo modelo.

RF-29 Listar Modelos Virtuales.

Descripción: El sistema debe permitir visualizar la respuesta de la petición realizada, mostrando una lista de los modelos existentes en el sistema.

Salida: listado de los modelos.

RF-30 Buscar Modelos Virtuales.

Descripción: El sistema busca en la base de datos del servidor el modelo especificado por el parámetro enviado y lo devuelve, sino devuelve la excepción en la respuesta.

Salida: Muestra el modelo según los datos pasados por parámetros.

RF-31 Exportar Modelos Virtuales.

Descripción: A partir de lo recibido por parámetros con los datos entrados por el usuario, el sistema debe permitir la opción de exportar el modelo especificado por los parámetros en formato Excel.

Descripción: El modelo es seleccionado por el usuario y luego es exportado al formato XML.

Entrada: selección del modelo que se desea exportar.

RF-32 Importar Modelos Virtuales.

Descripción: El usuario selecciona el modelo y luego son importados en formato XML.

Entrada: selección del archivo XML.

RF-33 Generar Rutina.

Descripción: Salva e Instala el corte como rutina.

Salida: Rutina guardada en la Base de datos.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con la toda la funcionalidad requerida, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación (JACOBSON, y otros, 2004).

Usabilidad:

RNF-1. Finalidad: El objetivo que persigue la aplicación es que los usuarios puedan tener una herramienta que les permita realizar funciones parametrizadas de manera visual en el SAGI.

Requisitos del Software:

RNF-2 Host para instalar la aplicación: La máquina donde se instalará la aplicación debe cumplir con los siguientes requisitos de software:

Sistema Operativo (SO): GNU/Linux Ubuntu 12.04 o superior, Debían 7 GNU/Linux o superior.

Paquetes: apache2, php5, libapache2-mod-php5, php5-cli, php5-pgsql, php5-xsl, php5-gd, php-curl.

Usuario con privilegios de administración del SO.

Tener instalador PostgreSQL en su versión 8.4 o posterior.

Requisitos del Hardware.

RNF-3 Las PC clientes deben cumplir con los siguientes requisitos de hardware:

Ordenador Pentium IV o superior, con 3.0 GHz de velocidad de microprocesador.

Memoria RAM, mínimo 1GB.

Restricciones de Diseño e Implementación:

RNF-4: Lenguaje y marco de trabajo (framework) para el desarrollo del sistema del lado del servidor.

El módulo deberá ser implementado en el lenguaje de programación PHP en su versión 5. Como framework de desarrollo se usará Symfony 1.1.7 el cual propone una arquitectura modular entre capas: el modelo, la vista y el controlador.

RNF-5: Lenguaje y marco de trabajo para el desarrollo del sistema del lado del cliente.

Uno de los framework fundamentales en el desarrollo de la herramienta es ExtJS 3.4 el cual permite el diseño de interfaces visuales interactivas usando tecnologías como AJAX y permiten crear aplicaciones web enriquecidas.

Interfaz:

RNF-6 Las interfaces de usuario serán diseñadas a modo de aplicaciones RIA (Rich Internet Application) lo que permite a los usuarios contar con aplicaciones web muy similares a las aplicaciones de escritorios. Para lograr este objetivo se usará la librería Javascript, ExtJS, la cual conjuga una serie de componentes visuales que proveen funcionalidades que ayudan al diseño de este tipo de aplicaciones web con apariencia de escritorio.

2.3. Modelo de Casos de Uso del Sistema

El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. Un caso de uso representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un Caso de Uso es una unidad simple de trabajo significativo. El modelo de casos de uso está compuesto por actores, casos de uso y la relación que existe entre ellos (Sparx, 2000-2007).

Caso de Uso: Un caso de uso es una técnica de modelado usada para describir lo que debería hacer un sistema nuevo o lo que hace un sistema que ya existe. Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de un usuario, permiten definir los límites del sistema y las relaciones entre el sistema y el entorno (JACOBSON, y otros, 2004).

Actores del Sistema:

Tabla 1: Descripción de los actores del sistema.

Actor	Descripción
Especialista	El especialista es la persona autorizada a utilizar el módulo.

2.3.1. Diagrama de Casos de Uso del Sistema

Los diagramas de casos de uso documentan el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto, los casos de uso determinan los requisitos funcionales del sistema, es decir, representan las funciones que un sistema puede ejecutar.

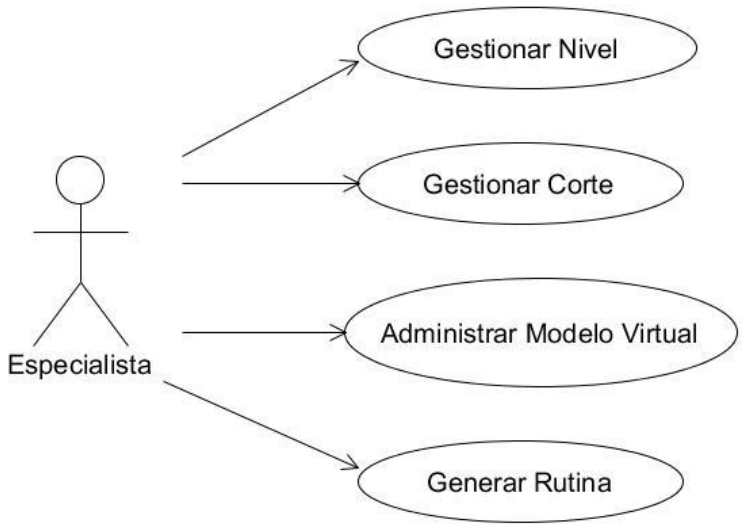


Fig. 2: Diagrama de caso de uso.

2.3.2. Patrones de casos de uso utilizados

Un patrón de casos de uso no describe un uso particular de un sistema. Más bien, captura las técnicas para que el modelo sea mantenible, reusable, y entendible. Entonces, podemos decir que los patrones de casos de uso capturan mejores prácticas para modelar casos de uso (JACOBSON, y otros, 2004).

CRUD Completo

Este patrón consta de un caso de uso, llamado ***Información CRUD*** (Creating, Reading, Updating, Deleting) o ***Gestionar información*** modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y estos a su vez son cortos y simples (JACOBSON, y otros, 2004).

CRUD Parcial

Este patrón alternativo modela una de las vías de los casos de uso como un caso de uso separado. Es preferiblemente utilizado cuando una de las alternativas de los casos de uso es más significativa, larga o más compleja que las otras (JACOBSON, y otros, 2004).

2.3.3. Descripción textual de los Casos de Uso del Sistema

Un caso de uso del negocio representa a un proceso de negocio, por lo que se corresponde con una secuencia de acciones que producen un resultado observable para ciertos actores del negocio. Desde la perspectiva de un actor individual, define un flujo de trabajo completo que produce resultados deseables (JACOBSON, y otros, 2004). A continuación, se muestra la descripción del caso de uso Gestionar Nivel (Tabla 2).

Tabla 2: Descripción del Caso de Uso Gestionar Nivel

Objetivo	Gestionar Nivel.
Actores	Especialista:(Inicia)
Resumen	El caso de uso comienza cuando el especialista accede al Módulo Generador de Rutina, a la opción Actualizar Nivel donde se realiza la gestión (Adiciona, Edita, Elimina, Lista y Filtra los niveles en el sistema.) de los mismos en el sistema. El caso de uso finaliza cuando lleva a cabo alguna de las acciones del caso de uso.
Complejidad	Alta
Prioridad	Alta
Precondiciones	Debe estar autenticado el especialista en el sistema y al menos debe de existir

	un modelo creado.	
Postcondiciones	Se gestionan los niveles en el sistema.	
Flujo de evento		
Flujo básico Gestionar nivel		
	Actor	**Sistema**
1.	Selecciona en el menú, la opción Actualizar Nivel.	
2.		Muestra la interfaz para la gestión de niveles la cual permite realizar las siguientes acciones: 1. Adicionar nivel. Ver Sección 1: Adicionar nivel. 2. Editar nivel. Ver Sección 2: Editar nivel 3. Eliminar nivel. Ver Sección 3: Eliminar nivel. 4. Buscar nivel. Ver Sección 4: Buscar nivel. 5. Listar nivel. Ver Sección 5: Listar nivel. 6. Exportar Nivel. Ver Sección 6: Exportar nivel. 7. Importar Nivel. Ver Sección 4: Importar nivel.
Sección 1: Adicionar nivel		
Flujo básico "Adicionar nivel"		
	Actor	**Sistema**
1.	Seleccionar la opción **Adicionar**.	
2.		Muestra la interfaz con los datos requeridos para la creación de un nuevo nivel. Los datos requeridos son: Nombre, Orden, SubStr, Descripción, Acumulado, Tipo de Origen, Clasificador, Tipo de Dato, Es maestro y Acepta columna.
3.	Llena los campos requeridos	
4.		Valida que los datos introducidos estén correctos y adiciona el nuevo corte a la lista de cortes y si el campo Es Maestro esta Validado, Actualizar Maestro, terminando así el caso de uso.
Flujo alterno		
3a Cancelar Adicionar nivel		

	Actor	Sistema
1.	Presionar el botón Cancelar.	
2.		Muestra la interfaz para la gestión de niveles, terminando así el caso de uso.
3b Los datos son incorrectos		
	Actor	**Sistema**
1.		Muestra un mensaje de error, indicando que los campos introducidos son incorrectos, terminando así el caso de uso.
4a El campo es maestro no está marcado		
	Actor	**Sistema**
1.		Valida que los datos introducidos estén correctos y adiciona el nuevo corte a la lista de cortes, terminando así el caso de uso, y no Actualizar Maestro
Sección 2: Editar nivel		
Flujo básico "Editar nivel"		
	Actor	**Sistema**
1.	Selecciona un nivel de la lista de niveles existentes y presiona la opción **Editar**	
2.		Muestra una interfaz con los campos requeridos para editar el nivel seleccionado.
3.	Introduce los datos que desea modificar del nivel seleccionado y presiona el botón **Aceptar**.	
4.		Actualiza el nivel, terminando así el caso de uso.
Flujo alterno		
5a Cancelar Editar nivel		
	Actor	**Sistema**
1.	Presionar el botón Cancelar.	
2.		Muestra la interfaz para la gestión de niveles, terminando así el caso de uso.

5b Los datos son incorrectos		
	Actor	**Sistema**
		Muestra un mensaje de error, indicando que los campos introducidos son incorrectos, terminando así el caso de uso.
Sección 3: Eliminar nivel		
Flujo básico "Eliminar nivel"		
	Actor	**Sistema**
1.	Selecciona un nivel de la lista de niveles existentes.	
2.		Resaltar el nivel seleccionado.
3.	Presionar la opción **Eliminar**.	
4.		El sistema muestra un mensaje para confirmar la acción de eliminar el nivel seleccionado.
5.	Seleccionar la opción **Si**.	
6.		Elimina el o los niveles de la lista de niveles y la actualiza, terminando así el caso de uso.
Flujo alterno		
5 No Eliminar nivel		
	Actor	**Sistema**
1.	Seleccionar la opción No.	
2.		Cierra el mensaje de confirmación de eliminar niel y muestra la interfaz para la gestión de niveles, terminando así el caso de uso.
Sección 4: Buscar nivel		
Flujo básico "Buscar nivel"		
	Actor	**Sistema**
1.		Muestra una ventana para introducir el criterio de búsqueda que puede ser el Nombre y Tipo de Origen.
2.	Introduce los criterios de búsqueda y presiona la opción **Buscar**.	

3.		Muestra los niveles que coincidan con el criterio de búsqueda introducido, terminando así el caso de uso.
Flujo alterno		
2 Criterios de búsqueda no válidos		
	Actor	**Sistema**
1.		Muestra la lista de niveles vacía, terminando así el caso de uso.
Sección 5: Listar nivel		
Flujo básico "Listar nivel"		
	Actor	**Sistema**
1.	Selecciona en el menú la opción Nivel.	
2.		Muestra una ventana donde se listan todos los niveles del sistema, terminando así el caso de uso.
Sección 6: Exportar nivel		
Flujo básico "Exportar nivel"		
	Actor	**Sistema**
1.	Selecciona un nivel de la lista de niveles existentes y presiona la opción **Editar**	
2.		Muestra una interfaz con los campos requeridos para exportar el nivel seleccionado. Los campos son: formato a exportar y dirección
3.	Especifica el formato y la dirección en la que desee guardar el nivel seleccionado y presiona el botón **Aceptar**.	
4.		Exporta el nivel, terminando así el caso de uso.
Flujo alterno		
3 Cancelar Exportar nivel		
	Actor	**Sistema**

1.	Seleccionar la opción Cancelar	
2.		Cierra la interfaz de exportar nivel y muestra la interfaz para la gestión de niveles, terminando así el caso de uso.
Sección 7: Importar nivel		
Flujo básico "Importar nivel"		
	Actor	**Sistema**
1.	Seleccionar la opción **Importar**.	
2.		El sistema muestra una interfaz para buscar los niveles a Importar
3.	Especifica la dirección del nivel deseado y selecciona el nivel a importar	
4.		Valida que el nivel seleccionado esté correcto y adiciona el nuevo nivel a la lista de niveles, terminando así el caso de uso.
Flujo alterno		
3 Cancelar Importar nivel		
	Actor	**Sistema**
1.	Seleccionar la opción Cancelar	
2.		Cierra la interfaz de Importar nivel y muestra la interfaz para la gestión de niveles, terminando así el caso de uso.

2.4 Diseño del sistema

En el diseño se modela el sistema y se define su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Además, impone una estructura del sistema (BRUEGGE, 2008).

Concretamente se define como **propósitos del diseño** (Larman, 2008):

- Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnologías de interfaz de usuario.
- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.

- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.
- Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software, lo cual es muy útil cuando utilizamos interfaces como elementos de sincronización entre diferentes equipos de desarrollo.

2.4.1. Modelo de diseño

El Modelo de Diseño es un modelo de objetos que describe la realización de los casos de uso y al mismo tiempo constituye una abstracción del modelo de implementación y del código fuente, constituye una entrada esencial a las actividades de implementación y prueba. El Modelo de Diseño constituye un vehículo de análisis durante la fase de elaboración, pero se refina con un diseño detallado durante la fase de construcción (BRUEGGE, 2008).

2.4.2. Patrones utilizados en el diseño de la solución

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Está compuesto por Clases, Instancias, Roles, Colaboraciones y la Distribución de Responsabilidades (Larman, 2008).

El patrón de diseño es (Larman, 2008):

- Una solución estándar para un problema común de programación.
- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Un proyecto o estructura de implementación que logra una finalidad determinada.
- Un lenguaje de programación de alto nivel.
- Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- Conexiones entre componentes de programas.

Patrón arquitectónico: El módulo se desarrollará en Symfony lo que implica que se utilice el patrón arquitectónico Modelo-Vista-Controlador (MVC). Symfony está basado en un patrón clásico del diseño web conocido como arquitectura MVC, que está formado por tres niveles:

El Modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.

La Vista transforma el modelo en una página web que permite al usuario interactuar con ella.

El Controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo

y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación (Potencier, 2015).

2.4.3. Diagrama de clases de diseño

El Diagrama de Clases de Diseño describe gráficamente las especificaciones de las Clases de Software y las Interfaces en una aplicación. Es una representación concreta de lo que se debe implementar. Estos diagramas representan la parte estática del sistema a través de la representación de las clases y sus relaciones (BOOCH, y otros, 2009). Los componentes de Symfony que intervienen en la construcción de la aplicación, al ser transparentes al programador se decidió modelarlos como un subsistema que recibió el nombre Symfony; además de la existencia de otro componente fundamental de Symfony, el cual se encarga de realizar el mapeo relacional de los objetos, el cual se denomina Propel. A continuación, se muestra un ejemplo (Fig. 3):

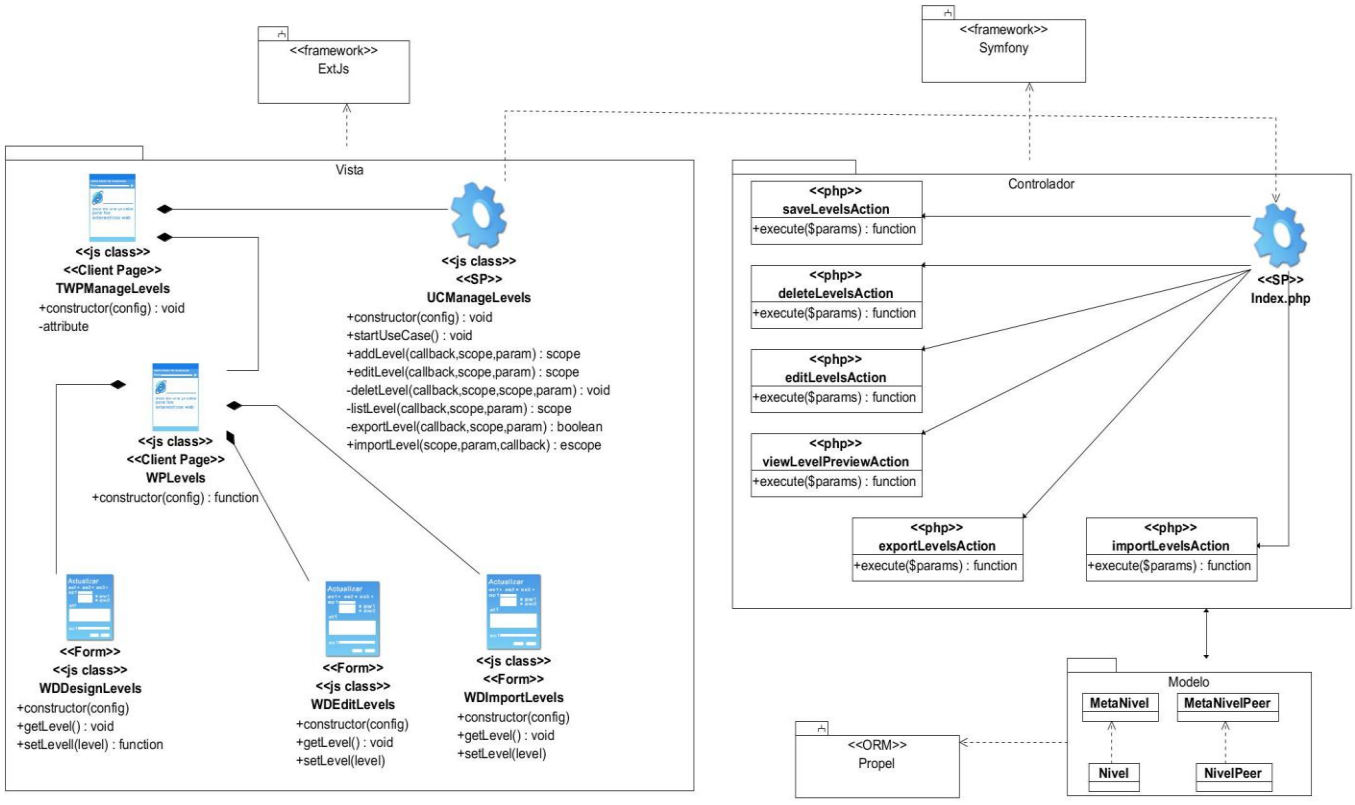


Fig. 3: Diagrama de clase de diseño del CU Gestionar Nivel.

Los diagramas de clases de diseño representan todas las clases con los métodos y atributos correspondientes a los casos de uso (CU). Para simplificar el desarrollo del módulo se utilizó la

automatización de uno de los patrones más utilizados en el framework Symfony: el patrón MVC. En este diagrama, los elementos del modelo corresponden a las clases generadas por el ORM Propel. El mismo crea cuatro clases por cada tabla de la base de datos. Las acciones, elementos que construyen las respuestas del servidor, están implementadas en el lenguaje PHP, con la dependencia del subsistema Symfony y comprobadas por el controlador frontal "index.php". Los elementos del lado del cliente corresponden a componentes específicos del negocio, en lenguaje JavaScript utilizando los componentes del subsistema ExtJS. A continuación, se describen las clases del caso de uso Gestionar Nivel.

En la vista se encuentran:

- ✓ La clase **UCManageLevels** es la encargada de gestionar todas las acciones del lado del cliente para para hacer llamadas a los diversos métodos en el lado del servidor.
- ✓ La clase **TWPManageLevels.js** es la encargada de construir la interfaz visual del módulo Actualizar Nivel utilizando la clase WPLevels.js.
- ✓ La clase **WPLevels.js** es la encargada de hacer llamadas a las clases WDDesignLevels, WDEditLevels, WDImportLevels.
- ✓ La clase **WDDesignLevels.**js es la que permite que se introduzcan los datos para adicionar el nivel.
- ✓ La clase **WDEditLevels.js** es la que permite que se introduzcan los datos para editar el nivel.
- ✓ La clase **WDImportLevels.js** es la que permite que se importen el o los niveles en el sistema.

En el modelo se encuentran las clases:

- ✓ Las clases generadas por Propel las cuales son las encargadas del acceso a los datos de la base de datos.

En el controlador se encuentran:

- ✓ La clase **index.php** es la encargada de responder las peticiones realizadas del lado del cliente a través de las clases tales como saveLevelsAction(), editLevelsAction(), deleteLevelsAction(), viewLevelsPreviewAction(), exportLevelsAction() e importLevelsAction().

Además, quedaron representados los **subsistemas:**

- ✓ El subsistema **ExtJS** es para atender las peticiones del lado del cliente.
- ✓ El subsistema **Symfony** es para atender las peticiones del lado del servidor.
- ✓ El subsistema **Propel** es para el manejo de los datos y la comunicación con los distintos gestores de bases de datos.

Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades) El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos (GAMMA, y otros, 2009).

Experto: Asignar una responsabilidad al experto en información, la clase que tiene la información necesaria para realizar la responsabilidad. Este es uno de los más utilizados, puesto que Propel es la librería externa que utiliza Symfony para realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades. Propel como ORM crea cuatro clases: dos clases de acceso a datos que trabajan directamente con la base de datos y dos de abstracción de datos que son las que tienen los atributos necesarios para realizar dicha función (Fig.4).

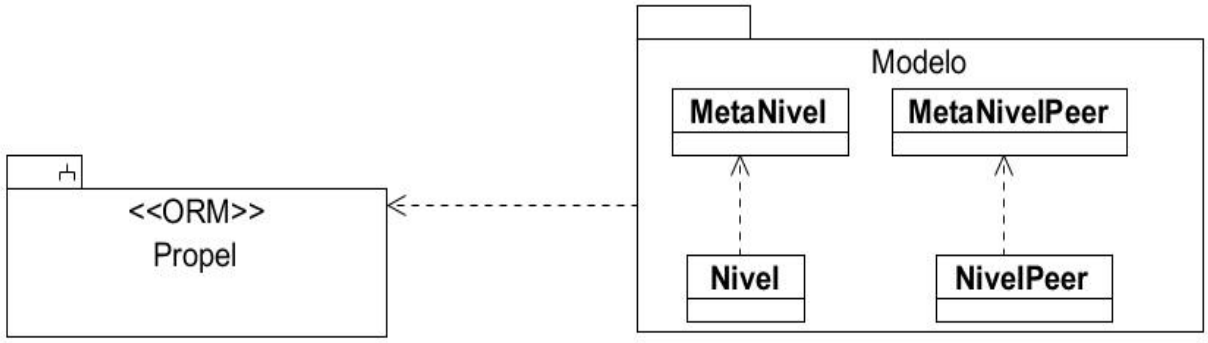


Fig. 4: Patrón Experto

Bajo Acoplamiento: Es la medida de cuánto una clase está conectada (tiene conocimiento) de otras clases. Es un patrón evaluativo: un bajo acoplamiento permite que el diseño de clases sea más independiente. La clase Actions hereda únicamente de sfActions para alcanzar un bajo acoplamiento de clases. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja (Fig.5).

```php
class addLevelsAction extends MySIGEAction {

    public function execute($request) {
        try {

            $level = new metaNivel();
            $level->setNombre($request->getParameter('nombre'));
            $level->setOrden($request->getParameter('orden'));
            $level->setSubstr($request->getParameter('substr'));
            $level->setLongstr($request->getParameter('longstr'));
            $level->setDescripcion($request->getParameter('descripcion'));
            $level->setAcumuladoNombre($request->getParameter('acumulado'));
            $level->setEnMaestro($request->getParameter('en_maestro'));
            $level->setAceptaColumna($request->getParameter('acepta_columna'));
            $level->setTorigen($request->getParameter('torigen'));
            $level->setClasificadorId($request->getParameter('codclasificador'));
            $level->setTdato($request->getParameter('tdato'));




            $level->save();

            return $this->renderText("{success:true}");
        } catch (Exception $e) {
            if ($e->getCode() == 1)
                return $this->renderText("{success:false, errors:{message:'" . $e->getMessage() .
            else {
```

Fig. 5: Patrón Bajo Acoplamiento.

Alta Cohesión: Define lo relacionadas que están las responsabilidades de una clase, o una clase con responsabilidades altamente relacionadas y que no lleva a cabo gran cantidad de trabajo. La clase Index.php interactúa con las 6 clases Actions para recuperar, crear y almacenar objetos (Fig.6).

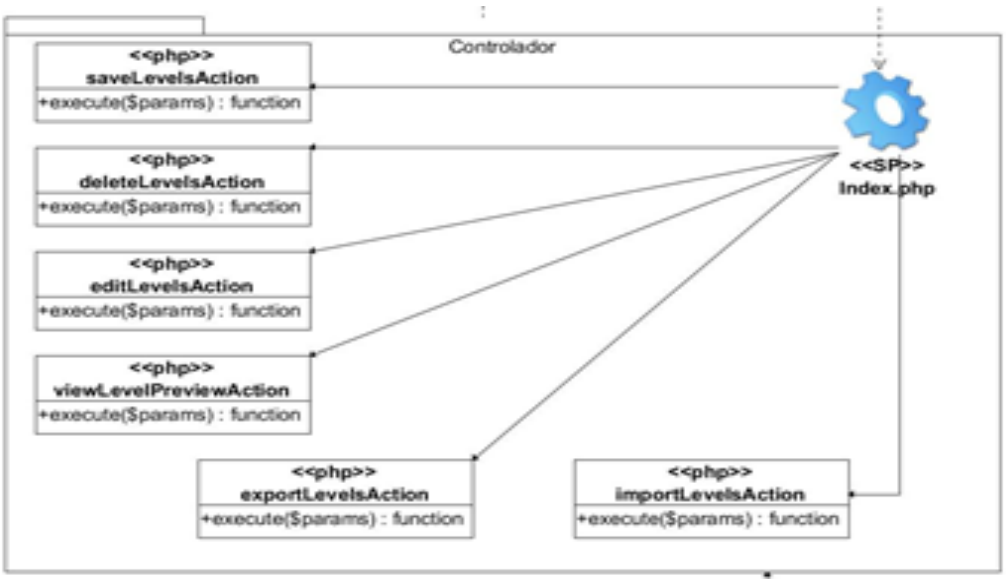


Fig. 6: Patrón Alta Cohesión.

Controlador: Es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. Un Controlador define el método para la operación del sistema. Todas las peticiones Web son manipuladas por un solo controlador frontal, que es el punto de entrada único de toda la aplicación en un entorno determinado. En este caso el "index.php" escucha las peticiones que vienen desde una URL, luego maneja la acción requerida para satisfacer la petición realizada (Fig.7).

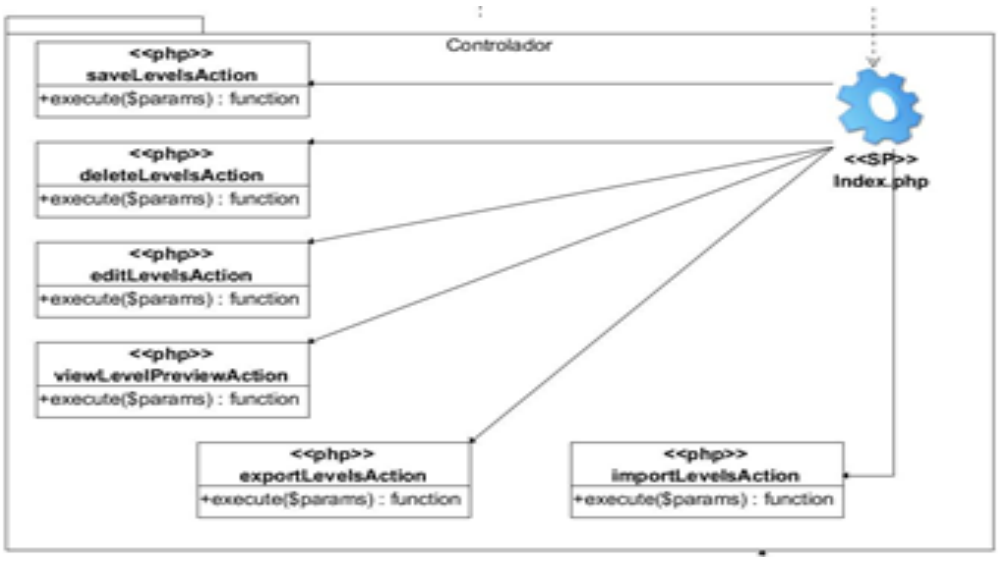


Fig. 7: Patrón Controlador.

Patrones GoF

Los patrones GoF (Gang of Four) se utilizan en situaciones frecuentes. Debido a que se basan en la experiencia acumulada al resolver problemas reiterativos. Ayudan a construir software basado en la reutilización, a construir clases reutilizables (JACOBSON, y otros, 2004).

Fachada (estructural): Se requiere una interfaz común para un conjunto de implementaciones o interfaces dispares. Se define la clase UCManageLevel como el único punto de conexión de la clase TWPManageLevel, este objeto fachada presenta una única interfaz y es responsable de colaborar con los clientes (Controlador de fachada) (Fig.8).

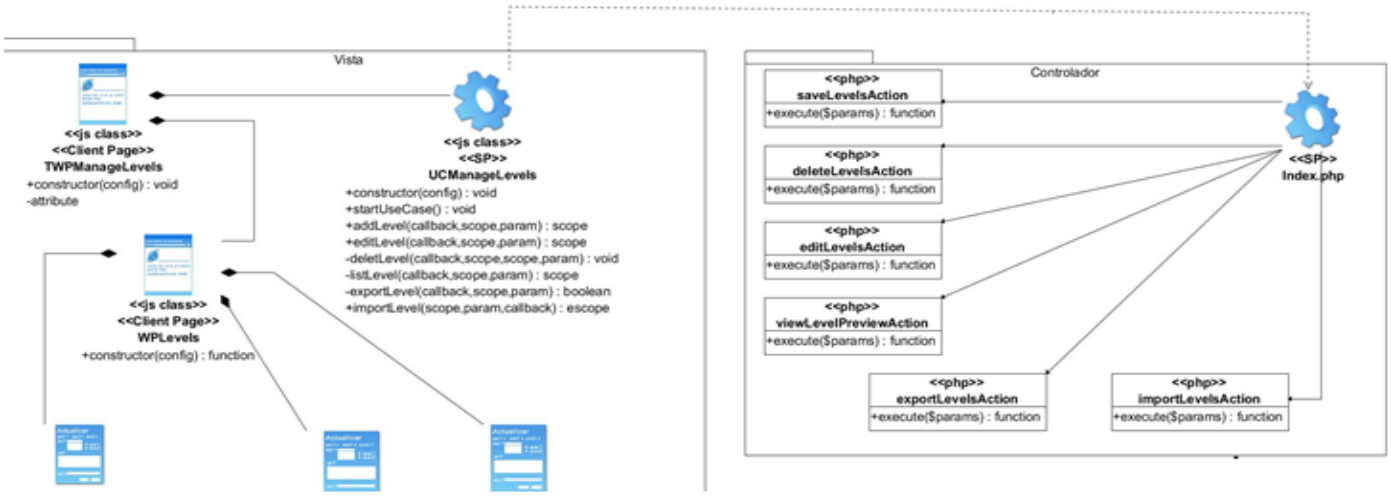


***Fig. 8:* Patrón Fachada.**

2.4.4. Diagrama de secuencia

Un diagrama de secuencia destaca el orden temporal de los mensajes. un diagrama de secuencia se forma colocando en primer lugar los objetos que participan en la interacción en la parte superior del diagrama, a lo largo del eje X. Normalmente, se coloca a la izquierda el objeto que inicia la interacción y los objetos subordinados a la derecha. A continuación, se colocan los mensajes que estos objetos envían y reciben a lo largo del eje Y, en orden de sucesión en el tiempo, desde arriba hasta abajo. Esto ofrece al lector una señal visual clara del flujo de control a lo largo del tiempo (BOOCH, y otros, 2009)(Fig.9).

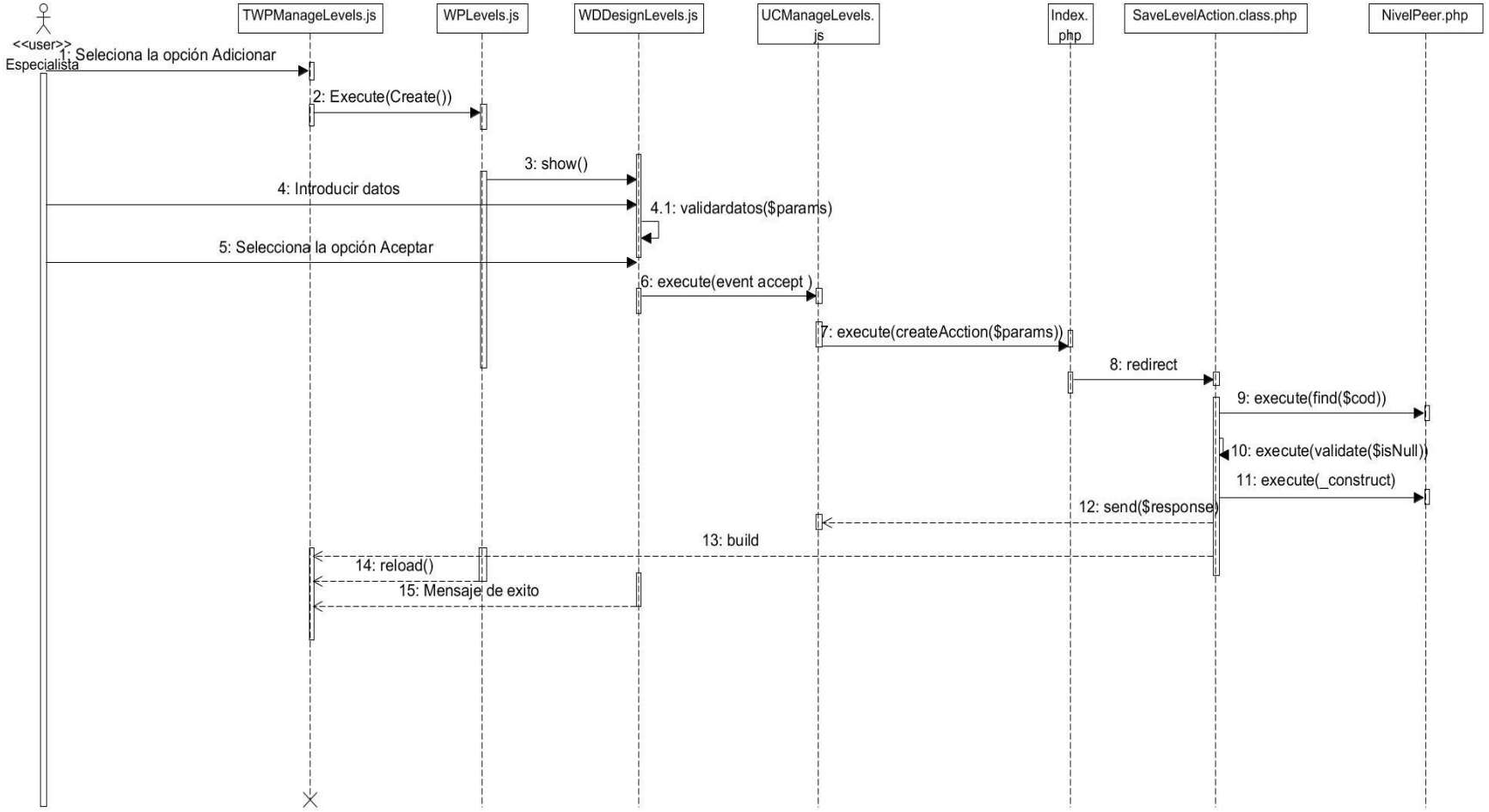


Fig. 9: Diagrama secuencia correspondiente al escenario Adicionar nivel.

En el diagrama de secuencia anterior el actor Especialista selecciona la opción Adicionar en la pantalla mostrada por la clase TWPManageLevel. Este ejecuta la acción create de la clase WPLevels, el cual realiza una llamada para que se muestre el WDNewLevel que le permite al especialista introducir los datos que se adicionarán. El especialista introduce los datos en el WDNewLevel, estos se validan y el especialista selecciona la opción Aceptar. Una vez aceptado se avisa al UCManageLevel del evento ocurrido. Este a su vez utiliza el record entregado por el WDNewLevel para realizar una petición sobre el controlador Index.php, el cual redirección la acción al controlador saveLevelAtion. El controlador realiza una búsqueda en el modelo para ver el código del nivel, luego ejecuta el método isNull para verificar que no exista un nivel con el mismo código y una vez comprobado crea un objeto de tipo NivelPeer. Al terminar la acción el controlador saveLevelAction envía una respuesta al UCManageNivel. Este construye el nuevo nivel. Luego la clase Factory realiza una llamada de recarga al TWPManageLevel actualizando de este modo el listado de niveles mostrados en este. Y por último el WDNewLevel envía un mensaje de éxito al TWPManageLevel.

2.4.5. Diagrama entidad-relación

Un diagrama o modelo entidad-relación es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información, así como sus interrelaciones y propiedades (JACOBSON, y otros, 2004)(Fig.10).

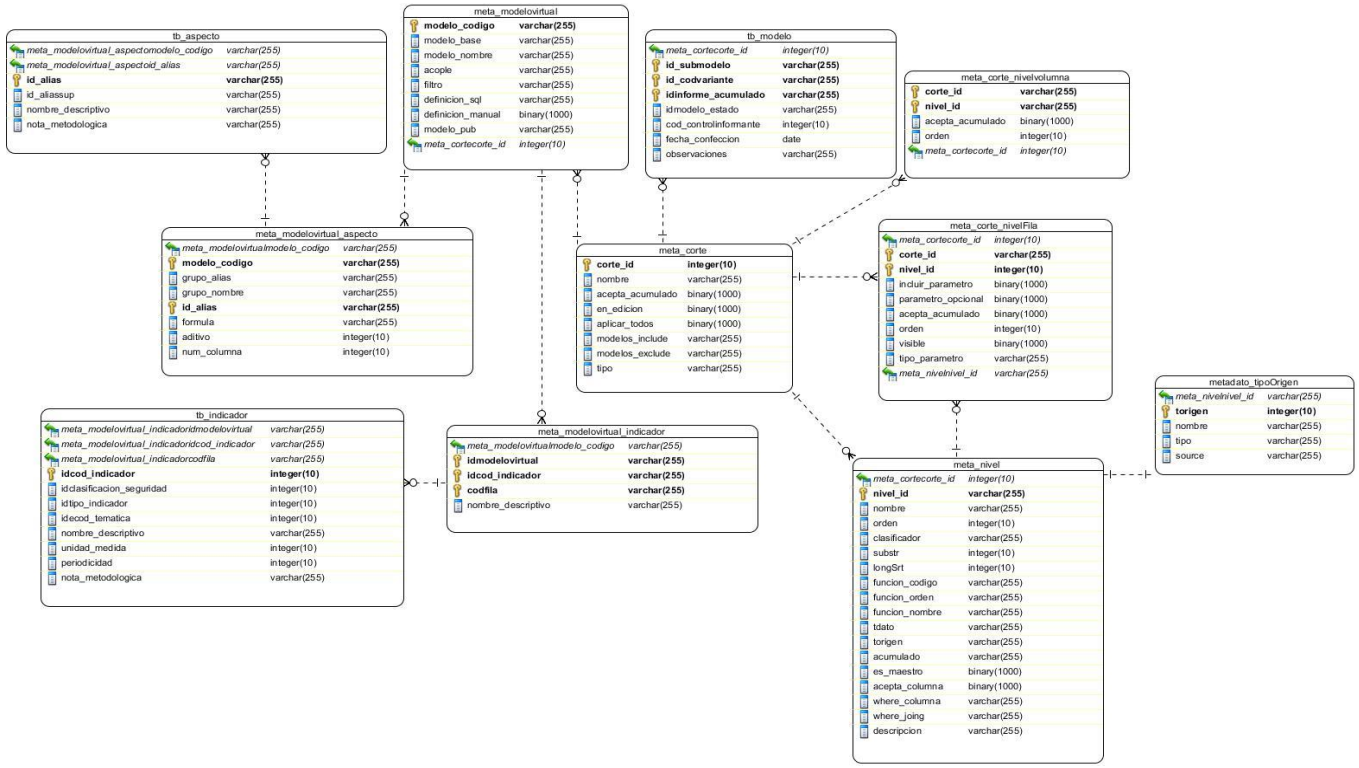


***Fig. 10:** Diagrama entidad-relación.*

2.4.6. Modelo de despliegue

Un Diagrama de Despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos (PRESSMAN, 2009)(Fig.11).

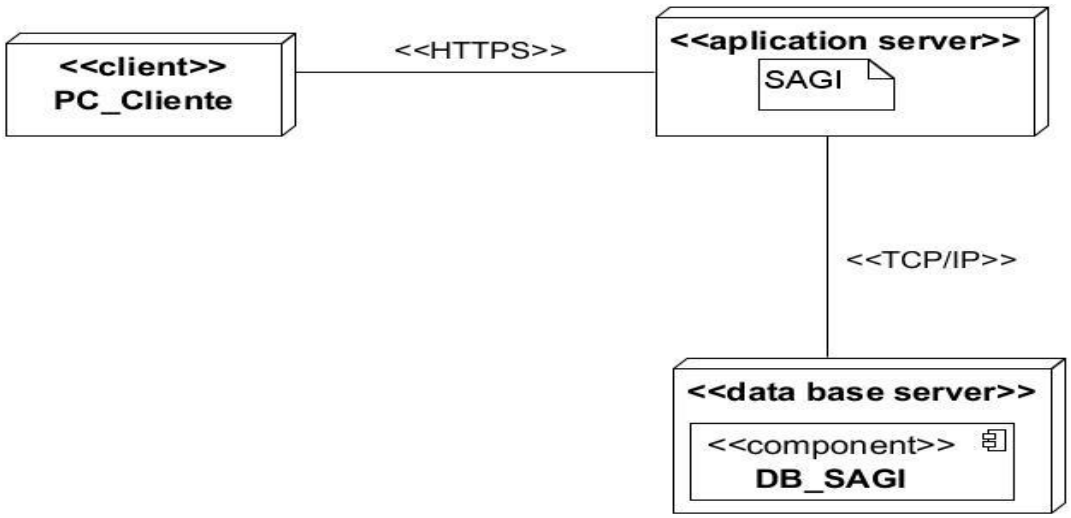


Fig. 11: Diagrama de Despliegue.

PC Cliente: se refiere a las estaciones de trabajo que el usuario utilizará para acceder a la aplicación Web y transcribir sus datos.

Servidor de Aplicación: servidor de aplicación utilizado para la publicación de la aplicación; y para lograr la conexión del sistema con la PC Cliente se utiliza HTTPS (Hypertext Transfer Protocol Secure) como protocolo de comunicación. Es la herramienta principal para ejecutar la lógica de negocio en el lado del servidor. Es el responsable de ejecutar el código de las páginas servidor. Se utiliza el servidor de aplicación.

Conexión HTTPS: es el protocolo utilizado entre el navegador de los clientes y el servidor Web. Este elemento de la arquitectura representa un tipo de comunicación no orientado a la conexión entre clientes y servidor.

Conexión TCP/IP: es la base del Internet que sirve para enlazar computadoras. El protocolo TCP/IP es utilizado para establecer la conexión entre el servidor de aplicación y el servidor de base de datos.

Servidor de Base de Datos: se refiere a un servidor que radica en cada nodo regional en el cual el cliente define que sean guardados los datos. En el servidor central estarán almacenados todos los datos recopilados por todos los nodos regionales. Los servidores de bases de datos elegidos son PostgreSQL, el cual está disponible para Linux y Windows.

Conclusiones del Capítulo

En el capítulo que finaliza se obtuvo un mayor entendimiento del negocio, donde se confeccionó el Modelo de Dominio. Se definieron los requisitos funcionales y no funcionales a tener en cuenta obteniendo la idea general de las funcionalidades que debe cumplir el sistema. Se identificaron 33 requisitos funcionales agrupados en 4 casos de usos del sistema, los cuales fueron relacionados mediante un diagrama de casos de uso del sistema y se les realizó una descripción detallada logrando así un mayor acercamiento a lo que el sistema deberá cumplir. Se determinó el patrón arquitectónico a seguir, así como los patrones de diseños utilizados en la realización de los diagramas de clases del diseño. Se generaron los diagramas de clases del diseño los cuales representan la estructura estática del sistema. Así como la generación de diagramas de secuencias para brindar una visión de cómo el usuario interactúa con la aplicación. A partir del diseño de clases persistentes se obtuvo el modelo de datos, donde este describe la representación lógica y física de los datos persistentes y la realización del diagrama de despliegue propició una visión de cómo está distribuido el sistema físicamente.

Capítulo 3: Implementación y Pruebas del Generador de Rutinas para SAGI.

Introducción

Como resultado de estas fases se obtiene un producto funcional para entregar a los clientes, con el que debe garantizarse el cumplimiento de los requisitos estipulados con la calidad requerida. En el capítulo se aborda sobre las actividades que se llevan a cabo durante la fase de implementación y pruebas. Se analiza el modelo de Implementación, así como una descripción de cómo los elementos del modelo de diseño se implementan en términos de componentes. Además, se diseña y aplican las pruebas para comprobar el correcto funcionamiento del módulo.

3.1. Modelo de implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes podemos encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos (PRESSMAN, 2009).

3.1.1. Diagrama de componentes

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación, especificando:

• Los subsistemas de implementación y sus dependencias a la hora de importar código.

• Organizar los subsistemas de implementación en capas.

También se utilizan para mostrar las dependencias de compilación de los ficheros de código, relaciones de derivación entre ficheros de código fuente y ficheros que son resultados de la compilación, dependencias entre elementos de implementación y los correspondientes elementos de diseños que son implementados (BOOCH, y otros, 2009)(Fig.12).

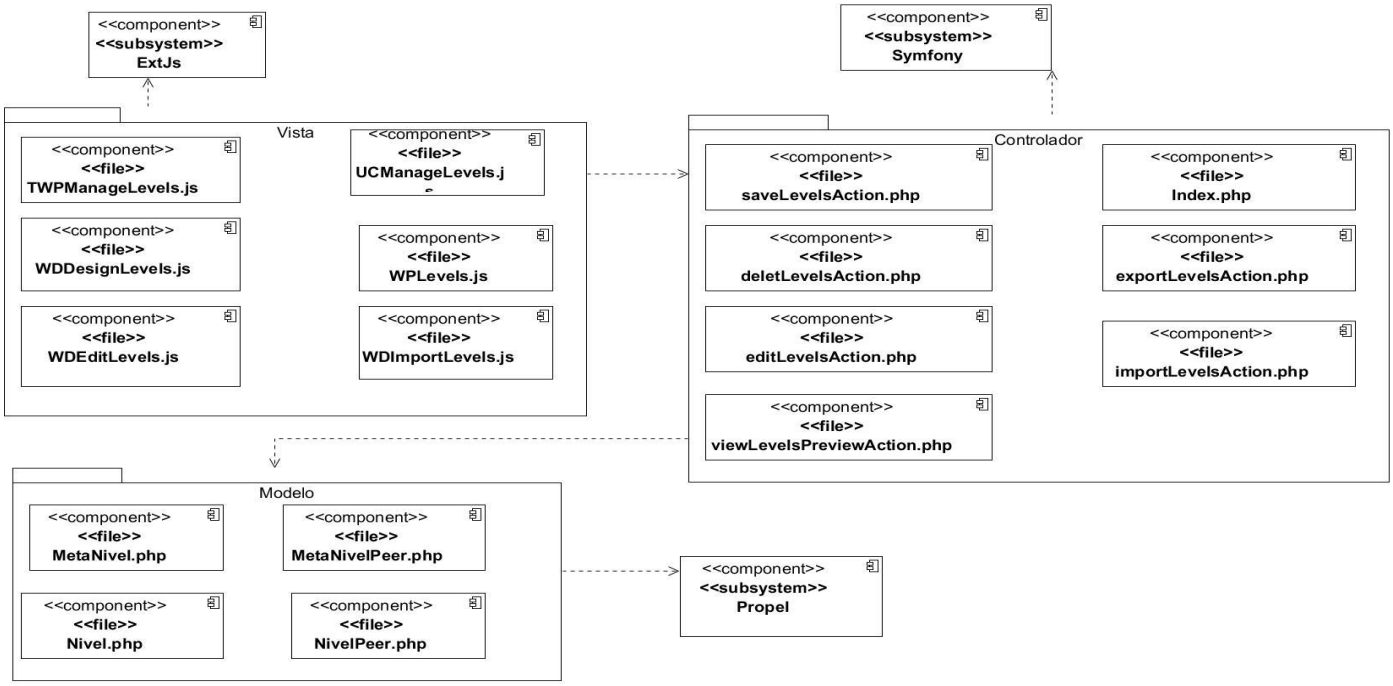


Fig. 12: Diagrama de Componente.

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos. Para simplificar el desarrollo del módulo se utilizó la automatización del patrón utilizado en el framework Symfony: el patrón MVC. Donde los elementos del lado del cliente están almacenados en el paquete vista utilizando los componentes del subsistema ExtJS. De esta manera en el paquete vista están agrupados 6 componentes que son los encargados de controlar todo el flujo de las interfaces, en este la clase UCManageLevel.js controla las principales acciones de llamada y retorno en la vista aceptando las peticiones de las demás clases(componentes) y generando el código Ajax pertinente para realizar las llamadas al Index.php, ubicada en el paquete controlador, donde las acciones, elementos que construyen las respuestas del servidor, están implementadas en el lenguaje PHP, con la dependencia del subsistema Symfony. Los elementos del modelo corresponden a las clases generadas por el ORM Propel. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja.

3.2. Código fuente

El código fuente es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está escrito por completo su funcionamiento. Estas líneas de texto están escritas en un lenguaje de programación específico y que puede ser leído por un programador. Debe traducirse a lenguaje máquina para que pueda ser ejecutado por la

computadora o a bytecode para que pueda ser ejecutado por un intérprete. Este proceso se denomina compilación (BOOCH, y otros, 2009).

3.2.1. Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, este debe tender siempre a lo práctico. Al comenzar un proyecto de software, se debe establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Lo cual permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia presente mantenibilidad (BRUEGGE, 2008).

3.2.2 Estándares de codificación utilizados

- Los bloques de código siempre deben estar encerrados por llaves, si consta de una línea no es necesario utilizar llaves.
- Los nombres de clases pueden contener sólo caracteres alfanuméricos.
- Los números están permitidos en los nombres de clase, pero desaconsejados en la mayoría de casos.
- Los nombres de funciones y variables deben empezar siempre con una letra minúscula utilizando la forma "camelCase".
- Se recomienda en sentido general la elocuencia, los nombres de las funciones deben ser elocuentes como para describir su propósito y comportamiento.
- Los ficheros tienen que concluir con la extensión .js, y no deben incluir signos de puntuación excepto – (guión medio) o _ (guión bajo).
- Los métodos de los objetos deben ser nombrados utilizando la forma ‘camelCase ‘.
- Usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso sí parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.
- Las cadenas tienen que ser definidas utilizando comillas simples siempre que sea posible, para obtener un mejor rendimiento.

A continuación, se muestra un ejemplo de estándares utilizados en el desarrollo de la aplicación:

```
public function execute($request) {
    try {
        $criteria = new Criteria();
        $modelos = ModeloPeer::doSelect($criteria);
        $result = array();
        foreach ($modelos as $modelo) {
            $temporal = array();
            $temporal['idnummodelo'] = $modelo->getIdnummodelo();
            $temporal['idsubnummodelo'] = $modelo->getIdsubnummodelo();
            $temporal['fechadeconfeccion'] = $modelo->getFechadeconfeccion();
            $temporal['value'] = $modelo->getIdnummodelo() . '' . $modelo->getIdsubnummodelo()
            $temporal['nameToShow'] = $modelo->getIdnummodelo() . '-' . $modelo->getIdsubnummc

            $result[] = $temporal;
        }


        $modelosvirtuales = metaModelovirtualPeer::doSelect($criteria);

        foreach ($modelosvirtuales as $virtual) {
            $temporal = array();
            $temporal['idnummodelo'] = $virtual->getModeloCodigo();
            $temporal['modelo_nombre'] = $virtual->getModeloNombre();
            $temporal['modelo_base'] = $virtual->getModeloBase();
```

Fig. 13: Código fuente.

3.3. Pruebas de software

Son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requisitos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente (PRESSMAN, 2009).

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. La creciente inclusión del software como un elemento más de muchos sistemas y la importancia de los costos asociados a un fallo del mismo, han motivado la creación de pruebas más minuciosas y bien planificadas.

Glem Myers establece varias normas que pueden servir adecuadamente como objetivos de la prueba:

1. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.

2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.

3. Una prueba tiene éxito si descubre un error no detectado hasta entonces. Los objetivos anteriores, suponen un cambio importante de punto de vista ya que la idea normal es que una prueba tiene éxito si no descubre errores. El objetivo de la prueba es: “es diseñar pruebas que saquen a la luz diferentes clases de errores con la menor cantidad de tiempo y espacio” (PRESSMAN, 2009).

Estrategia de Pruebas

La estrategia de prueba describe el enfoque y los objetivos generales de las actividades de prueba. Incluye los niveles de prueba (unidad, integración, etc.) a ser realizadas y el tipo de prueba a ser ejecutadas (funcional, stress, etc.) (PRESSMAN, 2009).

La estrategia define:

- Técnicas de pruebas y herramientas a ser usadas.
- Qué criterios de éxitos y culminación de la prueba serán usados.
- Consideraciones especiales afectadas por requisitos de recursos o que tengan implicaciones en la planificación.

Niveles de Pruebas

La Prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo.

Se distinguen los siguientes niveles de pruebas a realizar para evaluar el módulo:

- Prueba de Unidad
- Prueba de Integración
- Prueba de sistema
- Prueba de aceptación

3.3.1. Prueba 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 y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad siempre está orientada a caja blanca. Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente (PRESSMAN, 2009).

Método de Caja Blanca

Se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar (PRESSMAN, 2009).

Mediante la prueba de la caja blanca se puede obtener casos de prueba que:

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Es por ello que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican al *software*, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad (PRESSMAN, 2009).

Técnica 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* (PRESSMAN, 2009). 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 calcula la complejidad ciclomática del grafo.
- 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 derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Notación del Grafo de Flujo

Para aplicar la técnica del camino básico se debe introducir una sencilla notación para la representación del flujo de control, el cual puede representarse por un Grafo de Flujo. 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 ***(PRESSMAN, 2009)***.

```
public function execute($request) {
    try {
        $sql = $request->getParameter('sql');
        $codigo = $request->getParameter('codigo');
        $description = $request->getParameter('description');
        $modelo = $request->getParameter('modelo');
        $pagina = $request->getParameter('pagina');
        $aspectos = json_decode($request->getParameter('aspectos'));
        if (!empty($codigo) && !empty($modelo) && !empty($pagina) && !empty($aspectos)) {
            $mod = explode('-', $modelo);
            $mmv = new metaModelovirtual();
            $mmv->setModeloBase($mod[0] . '' . $mod[1]);
            $mmv->setModeloCodigo($codigo);
            $mmv->setModeloNombre($description);
            $mmv->setModeloPub($codigo);
            $mmv->save();

            foreach ($aspectos as $aspecto) {
                $asp = new metaModelovirtualAspecto();
                $asp->setModeloCodigo($codigo);
                $asp->setIdalias($aspecto->idalias);
                $asp->setNombredescriptivo($aspecto->nombredescriptivo);
                $numcol = PaginaaspectoPeer::getIncludeAspectsPage($mod[0], $mod[1], $pagina, $aspecto->idalias);
                $asp->setNumcolumna($numcol->getNumcolumna());
                $asp->save();
            }
        } else {
            return $this->renderText("{success:false, errors:{message: 'Imposible agregar el SubModelo, existen campos vacios'}}");
        }
        $conexion = Propel::getConnection();
        $consulta = strip_tags($sql);
        $sentencia = $conexion->prepareStatement($consulta);
        $sentencia->executeQuery();

        return $this->renderText("{success:true}");
    } catch (Exception $e) {
        $error = new Error("{success:false, errors:{message: 'Imposible guardar el submodelo.'}}", 1, $e);
        return $this->renderText($error->getMessage());
```

Fig. 14: Fragmento de código a analizar.

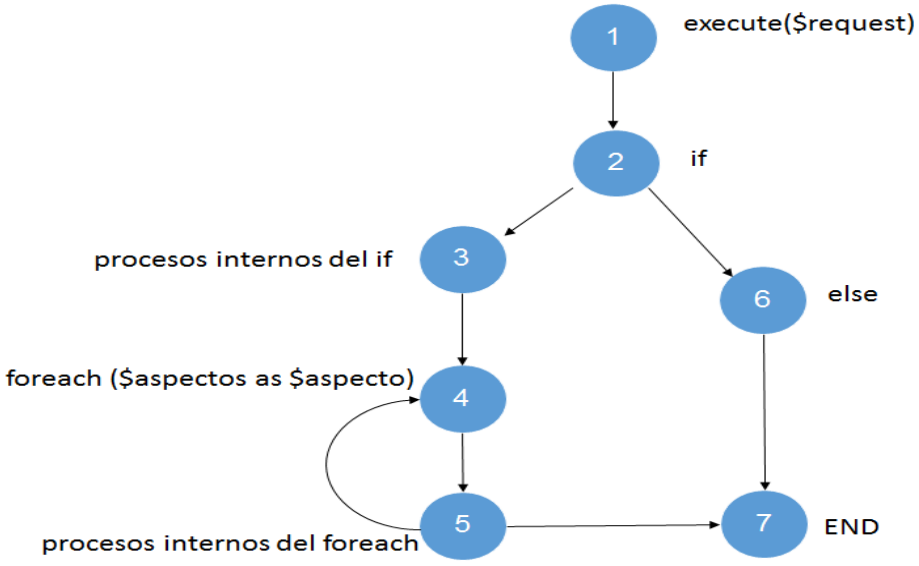


Fig. 15: Representación del grafo de flujo.

Complejidad ciclomática

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 camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente.

En la Figura 15 un ejemplo de camino independiente sería:

Camino 1: 1-2-3-4-5-7.

Camino 2: 1-2-6-7.

La complejidad ciclomática, *V(G)*, se define como:

V(G) = A - N + 2

Dónde: *A* es el número de aristas del grafo y *N* es el número de nodos.

V(G) = 8 aristas - 7 nodos + 2 = 3

Por tanto, la complejidad ciclomática del grafo de flujo de la figura 15 es 3.

Derivación de casos de Prueba

Luego de tener elaborados los Grafos de Flujos y los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino.

Luego de confeccionados los casos de prueba se ejecutaron cada uno de estos y se comparan los resultados con los esperados. Una vez terminados todos los casos de prueba, se está seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez.

3.3.2. Prueba de integración

Es ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso. Se prueba un paquete o un conjunto de paquetes del modelo de implementación. Estas pruebas descubren errores en las especificaciones de las interfaces de los paquetes. Esta prueba debe ser responsabilidad de desarrolladores y de independientes, sin solaparse las pruebas (PRESSMAN, 2009).

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. En este se verifica que módulos individuales son combinados y probados como un grupo. En una interfaz es posible perder datos, un módulo podría tener un efecto adverso e inadvertido sobre otro (PRESSMAN, 2009).

Se decide aplicar como tipo de prueba las pruebas de integración ascendente, como su nombre lo indica, empieza la construcción y la prueba con módulos atómicos (es decir, componentes de los niveles más bajos de la estructura del programa). Debido a que los componentes se integran de abajo hacia arriba, siempre

está disponible el procesamiento requerido para los componentes subordinados a un determinado nivel y se elimina la necesidad de resguardarlos.

Pasos para realizar la integración

- Creación de la app rutina por consola, así como los módulos manage_levels, manage_cuts y manage_VModel. Ej.:

  php symfony generate: app routine_generator
  php symfony generate: module routine generator manage levels
- En la base de datos con la que se trabaja se adiciona en la tabla napp el nombre de la clase que se creó y el nombre que aparecerá en la aplicación.
- En la tabla nmodule se ponen los nombres de los módulos creados, a que clase pertenecen y el nombre que tendrán en la aplicación.
- Se ejecuta el comando php symfony Propel: build-model para que Propel cree en el sistema las clases correspondientes a las tablas creadas en la BD.
- En la clase indexTestSuccess.php se agregan las rutas de los módulos.
- En la clase patdsi-base.js se especifican las rutas de las clases .js pertenecientes al módulo.
- Una vez realizados los pasos anteriores el módulo aparece en el sistema.

3.3.3. Pruebas del Sistema

Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. En un ciclo iterativo estas pruebas ocurren más temprano, tan pronto como subconjuntos bien formados de comportamiento de caso de uso son implementados (PRESSMAN, 2009).

Proporciona un aseguramiento final de que el software cumple con todos los requisitos funcionales, de comportamiento y desempeño. La prueba se concentra en las acciones visibles para el usuario y en la salida que este puede reconocer. La validación se alcanza cuando el software funciona de tal manera que satisface las expectativas razonables (especificación de requisitos de software) del cliente (PRESSMAN, 2009).

Se decide aplicar como tipo de prueba las pruebas funcionales ya que asegura el apropiado trabajo de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. Las pruebas funcionales deben enfocarse en los requisitos funcionales, las pruebas pueden estar basadas directamente en los CU y las reglas del negocio. Las metas de estas pruebas son verificar la apropiada aceptación de datos y verificar el procesamiento, recuperación e implementación adecuada de las reglas del negocio. Este tipo de pruebas están basadas en técnicas de caja negra, que es, verificar la

aplicación (y sus procesos internos) mediante la interacción con la aplicación y analizar la salida (resultados).

Caja Negra este método se centra en los requisitos funcionales del software. O sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (PRESSMAN, 2009).

La partición equivalente es una técnica de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El diseño de casos de prueba para partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada (PRESSMAN, 2009).

Tabla 3: Secciones de pruebas para el CU Gestionar Nivel

Nombre de la sección	Descripción de la funcionalidad
SC1 Adicionar Nivel	El especialista decide insertar un nuevo nivel, el sistema envía un mensaje indicando que la petición se realizó exitosamente, terminando así el CU.
SC2 Editar Nivel	El especialista decide editar un nivel existente, el sistema envía un mensaje indicando que la petición se realizó exitosamente, terminando así el CU.

Tabla 4: Caso de prueba Adicionar Nivel del CU Gestionar Nivel.

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5	Variable 6	Variable 7	Variable 8	Variable 9	Variable 10	Respuesta del sistema	Flujo central
EC 1.1 Adicionar nivel	*En este escenario se realiza la adición de un nuevo nivel al sistema correctamente.*	*V*	*V*	*V*	*V*	*V*	*V*	*V*	*V*	*V*	*V*	*Se adiciona correctamente el convenio y el sistema permite la inserción de uno nuevo. Muestra un mensaje confirmando el éxito de la operación.*	*1. Se llenan los campos correctamente.* *2. Se presiona el botón Aceptar.* *3. Se resetean los campos.*
		Empresa	*1*	*3*	*empresa*	*13*	*vm1234*	*mío*	*string*	*true*	*true*		
EC 1.2 Adicionar nivel con datos incorrectos	*En este escenario se realiza la adición de un nuevo nivel al sistema con datos incorrectos.*	*V*	*V*	*I*	*V*	*V*	*V*	*V*	*V*	*V*	*V*	*El sistema valida que los campos estén correctamente y muestra un mensaje informando que existen datos no válidos.*	*1. Se llenan los campos con datos no válidos.*
EC 1.3 Cancelar operación	*En este escenario se cancela la operación de adicionar un nuevo NIVEL*											*En este escenario se cancela la operación de adicionar un nuevo nivel.*	*2. Se completan los datos.* *3. Se presiona el botón Cancelar.*

Tabla 5: Caso de prueba Modificar Nivel del CU Gestionar Nivel.

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5	Variable 6	Variable 7	Variable 8	Variable 9	Variable 10	Respuesta del sistema	Flujo central
EC 1.1 Modificar nivel	En este escenario se realiza la edición de un nivel existente en el sistema correctamente.	V	V	V	V	V	V	V	V	V	V	Se edita correctamente el nivel seleccionado. El sistema muestra un mensaje confirmando el éxito de la operación.	1. Se editan los campos correctamente. 2. Se presiona el botón Aceptar.
					true	true	true						
EC 1.2 Modificar nivel con datos incorrectos	En este escenario se realiza la edición de un nivel existente en el sistema con datos incorrectos.	I	V	V	V	V	V	V	V	V	V	El sistema valida que los campos estén correctamente y muestra un mensaje informando que existen datos no válidos.	1. Se editan los campos con datos no válidos.
EC 1.3 Cancelar operación	En este escenario se cancela la operación de editar un nivel existente	I	V	V	V	V	V	V	V	V	V	El sistema muestra una notificación "Error al cambiar la propiedad nombre: Nivel Nuevo nivel 7 ya existe".	1. Seleccionar el nivel a modificar. 2. Nombrar el nivel con el mismo nombre de un nivel en existencia.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	campo de texto	No	Nombrar del nivel. Cadena. Admite A-Z, a-z
2	Orden	campo de texto	No	Código con que se identificará el nivel. Numérico. Admite 0-9
3	SubStr	campo de texto	No	En este campo se le permite al usuario poner un subtítulo al cubo creado.
4	Descripcion	campo de texto	Si	Descripción adicional del nivel. Cadena. Admite A-Z, a-z
5	Acumulado	campo de texto	Si	Monto total: Numérico. Admite 0-9
6	Tipo de Origen	menu desplegable	No	Identificador del modelo que procede
7	Clasificador	menu desplegable	No	Clasificación con que se muestra el nivel.
8	Tipo de Dato	menu desplegable	No	Tipo del dato
9	Es maestro	check box	Si	Define si el nivel procede como maestro
10	Acepta Columna	check box	Si	Define si el nivel acepta columna

Fig. 16: Descripción de las variables de caso de pruebas.

Tras aplicar las pruebas funcionales mediante el método de caja negra, utilizando los casos de prueba asociados a cada caso de uso, se comprobó el correcto funcionamiento del módulo generador de rutinas para SAGI. Durante las tres iteraciones realizadas, se detectaron un total de siete no conformidades, quedando todas resueltas, como se muestra sobre la figura 15 y la tabla 6.

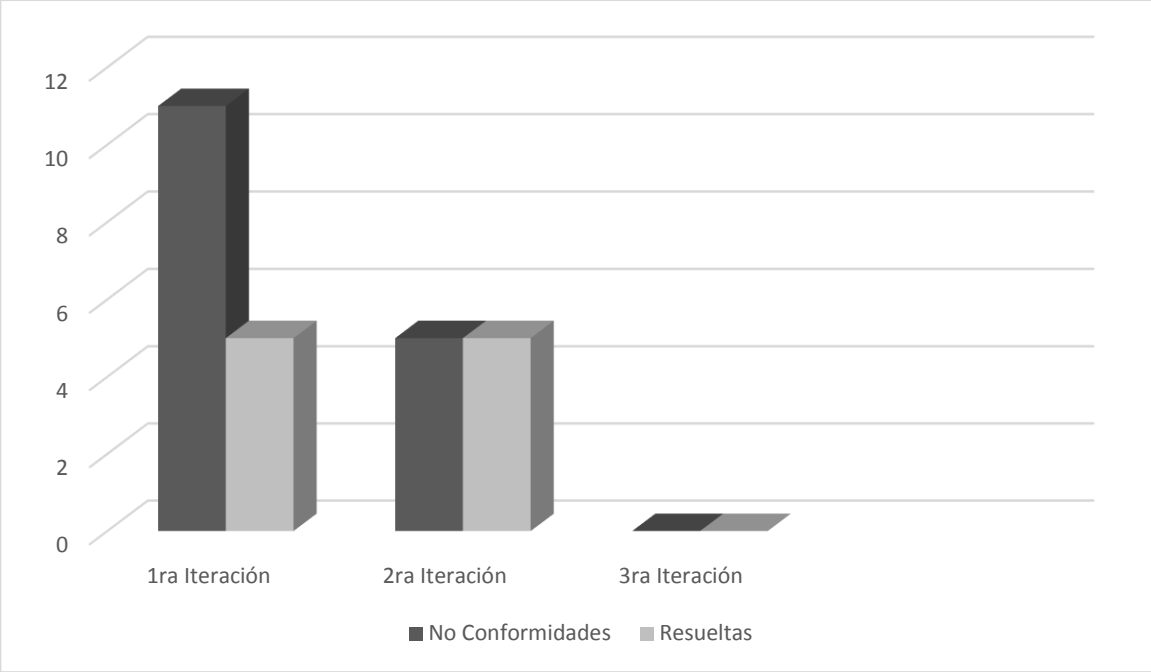


Fig.17: Gráfico que representa la relación de No Conformidades (NC) Detectadas y NC Corregidas.

Tabla 6: Descripción de las Iteraciones realizadas durante las pruebas.

No. Iteración	Cant.NC	Tipo Error	Impacto
1	11	4 Ortografía	Bajo
		2 Interfaz	Medio
		5 Validación	Alto
2	5	5 Validación	Alto
3	0	-	-

3.3.4 Prueba de aceptación

Prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido (PRESSMAN, 2009).

Las pruebas de aceptación son aquellas que son diseñadas por el propio equipo de desarrollo en base a los requisitos funcionales especificados, y ejecutadas por el propio usuario de manera que este dé validez y conformidad al producto que se les está entregado en base a lo que se acordó inicialmente. De forma general las pruebas de aceptación pueden afrontarse mediante dos tipos de procedimiento para realizarlas: pruebas alfa y pruebas beta (PRESSMAN, 2009).

Se decide aplicar las pruebas alfa ya que en ellas se le entrega a un usuario final todo el producto terminado, junto a su documentación correspondiente para que este, en presencia del desarrollador y en entornos previamente preparados para el proceso de dichas pruebas, vaya informando de las inconsistencias y errores que detecte. En este caso el usuario final será la ONEI el cual en conjunto con el líder del proyecto y los desarrolladores efectuaran las pruebas de aceptación, ver anexo 2.

Conclusiones del capítulo

En el desarrollo del presente capítulo se realizó el modelo de implementación del módulo con el propósito de mostrar los componentes del sistema y sus relaciones, a través del diagrama de componentes. Se especificó el uso de los estándares de codificación para lograr un estilo claro y organizado del código durante la fase de implementación. Para evaluar la calidad de la aplicación se validó la completitud de los requisitos con las pruebas funcionales al software. Se identificaron un total de 11 NC las cuales fueron resueltas paulatinamente, obteniéndose un producto libre de errores y listo para su ejecución. Se comprobó que el Módulo Generador de Rutinas para el Sistema Automatizado para la Gestión de la Información obtenido satisface las necesidades de la ONEI.

Conclusiones generales

- ✓ El análisis de SAGI y SIGETools, facilitó la selección de las herramientas y las tecnologías a utilizar para el correcto desarrollo del módulo.
- ✓ El análisis y obtención de los 33 requisitos funcionales y 7 no funcionales del Módulo para la Generación de Rutinas para el SAGI, permitió el correcto diseño de las clases mediante la utilización de patrones de diseño.
- ✓ La implementación de las funcionalidades del módulo permitió obtener una aplicación informática que responde a las necesidades del cliente definidas durante la etapa de análisis.
- ✓ Las pruebas diseñadas y aplicadas al módulo garantizaron el correcto funcionamiento del mismo, detectando no conformidades que fueron solucionadas posteriormente.
- ✓ El desarrollo del Módulo para la Gestión de Rutinas para el SAGI permitió la creación de las mismas en el SAGI posibilitando así un mayor control del cumplimiento de los reportes y que los mismos puedan ser entregados y analizados en el tiempo estimado.

Recomendaciones

Al término de esta investigación se recomienda:

- ✓ Incorporar el procedimiento de edición de las vistas creadas por los modelos virtuales para que a su vez estas sean reutilizadas. Con esta funcionalidad se optimiza el trabajo con los modelos virtuales.
- ✓ Permitir al módulo la realización de rutinas más complejas.
- ✓ Incluir el módulo generador de rutinas en las demás distribuciones del Sistema Integrado de Gestión Estadística.

Referencias Bibliográficas

Agualló, P. 2010. Desarrollo Cliente / servidor: ubicación de las reglas de negocio. *Desarrollo Cliente / servidor: ubicación de las reglas de negocio.* [En línea] 2010. [Citado el: 15 de febrero de 2015.] http://www.ctv.es /USERS/pagullo /arti/esbr/esbr.htm..

Anglada, Alain Abel Garófalo Hernández. 2013. Revista Cubana de Ciencias Informáticas. *Revista Cubana de Ciencias Informáticas.* [En línea] abr-jun. de 2013. [Citado el: 26 de octubre de 2015.] http://scielo.sld.cu/scielo.php?pid=S2227-18992013000200006&script=sci_arttext. ISSN 2227-1899.

Boehm. *SW Cost Estimation with COCOMO II.*

BOOCH, Grady, RUMBAUGH, James, JACOBSON y Ivar. 2009. *El lenguaje unificado de modelado. Manual de.* 2009.

Bruegge, B. Y Dutoit, A. 2002. *Ingeniería de Software Orientado a Objetos.* 2002.

BRUEGGE, Bernd, DUTOIT, Allen. 2008. *Ingeniería de software orientado a objetos.* 2008.

CHEN, Dao-xin. 2011. *EXTJS Framework in the Development of Application [J]. Computer Knowledge and Technology.* 2011.

Cornejo, José Enrique González. 2008. DocIRS. *DocIRS.* [En línea] DocIRS, enero de 2008. [Citado el: 24 de octubre de 2015.] http://www.docirs.com/uml.htm.

disca. 2012. disca. *disca.upv.* [En línea] upv, enero de 2012. http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

eclipse. 2011. eclipse. *eclipse.* [En línea] 2011. [Citado el: 10 de 7 de 2016.] http://epf.eclipse.org/wikis/openup/.

GAMMA, E., HELM, R. y JOHNSON, R. y VLISSIDES, J. 2009. Patrones de diseño. *Patrones de diseño.* [En línea] vico, 2009. http//www.vico.org/pages/PatronsDisseny.html..

genbetadev. 2014. genbetadev: Desarrollo de Software. *genbetadev: Desarrollo de Software.* [En línea] 9 de enero de 2014. [Citado el: 15 de diciembre de 2015.] http://www.genbetadev.com/herramientas/netbeans-1.

ibiblio. 2014. ibiblio. *ibiblio.org.* [En línea] Postgresql, 2014. https://www.ibiblio.org/pub/linux/docs/LuCaS/Postgresql-es/web/navegable/todopostgresql/xfunc.htm.

IBM. 2009. IBM Knowledge Center. *IBM Knowledge Center.* [En línea] IBM, diciembre de 2009. [Citado el: 24 de octubre de 2015.] /www-01.ibm.com/support/knowledgecenter/SSEPGG_8.2.0/com.ibm.db2.udb.doc/ad/c0009415.htm?lang=es.

INTECO, Laboratorio Nacional de Calidad del Software. 2010. *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA.* Gobierno de España : s.n., 2010.

JACOBSON, Ivar y BOOCH, Grady, RUMBAUGH, James. 2004. *El Proceso Unificado.* 2004.

KANEIWA, Ken, MIZOGUCHI, Riichiro y NGUYEN, Philip HP. 2015. *A Logical and Ontological Framework for Compositional Concepts of Objects and Properties.* s.l. : New Generation Computing, 2015.

Larman, C. 2008. *UML y patrones.* 2008.

Marin, Marvin David Arias. 2008. Catedra de Programación. *Catedra de Programación.* [En línea] 16 de octubre de 2008. [Citado el: 26 de octubre de 2015.] http://catedraprogramacion.forosactivos.net/t83-definicion-de-lenguaje-de-programacion-tipos-ejemplos.

MARTÍNEZ, Marilé Lemus, et al. 2016. *Sistema para la Informatización de la Gestión de Anuarios Estadísticos en la ONEI.* s.l. : Serie Científica-Universidad de las Ciencias Informáticas, 2016.

Martinez, Rafael. 2013. PostgreSQL-es. *PostgreSQL-es.* [En línea] 2013. [Citado el: 15 de diciembre de 2015.] http://www.postgresql.org.es/sobre_postgresql.

MDN. 2009. MDN. *MDN.* [En línea] 2009. [Citado el: 26 de octubre de 2015.] https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Introducci%C3%B3n.

MICHAUD, Frederic. 2015. *Identifying Key Attack Surface Resources with Dynamic Analysis.* 2015.

MORO, Alfonso Infante, PÉREZ, Oscar Gallego y MACÍAS, Amparo Sánchez. 2013. *Los gadgets en las plataformas de telefomación: el caso del proyecto DIPRO 2.0. Pixel-Bit: Revista de medios y educación, 2013.* 2013.

OBE, Regina O. y HSU, Leo S. 2015. *PostGIS in action.* s.l. : Manning Publications Co, 2015.

ÖVERGAARD, Gunnar y PALMKVIST, Karin,. 2008. *Use Cases: Patterns and Blueprints.* 2008.

Paradigm. 2010. Paradigm, 2010. *Paradigm, 2010.* [En línea] www.visual-paradigm.com, 2010. [Citado el: 15 de noviembre de 2015.] http://www.visual-paradigm.com/aboutus/newsreleases/vpuml80.jsp.

PAYER, Mathias, BARRESI, Antonio y GROSS, Thomas R. 2015. *Fine-grained control-flow integrity through binary hardening. En International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* s.l. : Springer International Publishing, 2015.

PHP. 2011. php. *php.* [En línea] php.net, 2011. [Citado el: 26 de octubre de 2015.] http://php.net/manual/es/preface.php.

Potencier, Fabien. 2015. librosweb. [En línea] 2015. [Citado el: 6 de Febrero de 2016.] http://librosweb.es/libro/symfony_1_1/capitulo_2/el_patron_mvc.html.

Potencier, François Zaninotto. 2009. Symfony, la guia definitiva. *Symfony, la guia definitiva.* [En línea] 2009. [Citado el: 20 de noviembre de 2015.] http://librosweb.es/libro/symfony_1_4/.

PRESSMAN, Roger. 2009. *Ingeniería del Software. Un enfoque práctico.* s.l. : McGraw-Hill/Interamericana, 2009.

prueba, Condiciones y casos de. Condiciones y casos de prueba. *Condiciones y casos de prueba.* [En línea] http://www.rmya.com.ar.

Reingart, Mariano. 2011. postgresql. *postgresql.* [En línea] 2011. [Citado el: 26 de octubre de 2015.] http://www.postgresql.org.ar/trac/wiki/PgAdmin?format=pdfarticle.

Sparx. 2000-2007. Sparx Systems. *Sparx Systems.* [En línea] Sparx Systems, 2000-2007. [Citado el: 21 de enero de 2016.] http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html. ISBN.

STEWART, Claire R., et al. 2016. *Sensory symptoms and processing of nonverbal auditory and visual stimuli in children with autism spectrum disorder.* s.l. : Journal of autism and developmental disorders, 2016.

uml-diagrams. 2010. uml-diagrams. *uml-diagrams.* [En línea] 2010. [Citado el: 14 de octubre de 2015.] http://www.uml-diagrams.org/.

Bibliografía

Agualló, P. 2010. Desarrollo Cliente / servidor: ubicación de las reglas de negocio. *Desarrollo Cliente / servidor: ubicación de las reglas de negocio.* [En línea] 2010. [Citado el: 15 de febrero de 2015.] http://www.ctv.es SERS/pagullo /arti/esbr/esbr.htm..

Anglada, Alain Abel Garófalo Hernández. 2013. Revista Cubana de Ciencias Informáticas. *Revista Cubana de Ciencias Informáticas.* [En línea] abr-jun. de 2013. [Citado el: 26 de octubre de 2015.] http://scielo.sld.cu/scielo.php?pid=S2227-18992013000200006&script=sci_arttext. ISSN 2227-1899.

Boehm. *SW Cost Estimation with COCOMO II.*

BOOCH, Grady, RUMBAUGH, James, JACOBSON y Ivar. 2009. *El lenguaje unificado de modelado. Manual de.* 2009.

Bruegge, B. Y Dutoit, A. 2002. *Ingeniería de Software Orientado a Objetos.* 2002.

BRUEGGE, Bernd, DUTOIT, Allen. 2008. *Ingeniería de software orientado a objetos.* 2008.

CHEN, Dao-xin. 2011. *EXTJS Framework in the Development of Application [J]. Computer Knowledge and Technology.* 2011.

Cornejo, José Enrique González. 2008. DocIRS. *DocIRS.* [En línea] DocIRS, enero de 2008. [Citado el: 24 de octubre de 2015.] http://www.docirs.com/uml.htm.

disca. 2012. disca. *disca.upv.* [En línea] upv, enero de 2012. http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

eclipse. 2011. eclipse. *eclipse.* [En línea] 2011. [Citado el: 10 de 7 de 2016.] http://epf.eclipse.org/wikis/openup/.

fergarciac. 2013. fergarciac. *fergarciac.* [En línea] 25 de enero de 2013. [Citado el: 14 de diciembre de 2015.] https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/.

GAMMA, E., HELM, R. y JOHNSON, R. y VLISSIDES, J. 2009. Patrones de diseño. *Patrones de diseño.* [En línea] vico, 2009. http//www.vico.org/pages/PatronsDisseny.html..

genbetadev. 2014. genbetadev: Desarrollo de Software. *genbetadev: Desarrollo de Software.* [En línea] 9 de enero de 2014. [Citado el: 15 de diciembre de 2015.] http://www.genbetadev.com/herramientas/netbeans-1.

ibiblio. 2014. ibiblio. *ibiblio.org.* [En línea] Postgresql, 2014. https://www.ibiblio.org/pub/linux/docs/LuCaS/Postgresql-es/web/navegable/todopostgresql/xfunc.htm.

IBM. 2009. IBM Knowledge Center. *IBM Knowledge Center.* [En línea] IBM, diciembre de 2009. [Citado el: 24 de octubre de 2015.] /www-01.ibm.com/support/knowledgecenter/SSEPGG_8.2.0/com.ibm.db2.udb.doc/ad/c0009415.htm?lang=es.

INTECO, Laboratorio Nacional de Calidad del Software. 2010. *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA.* Gobierno de España : s.n., 2010.

JACOBSON, Ivar y BOOCH, Grady, RUMBAUGH, James. 2004. *El Proceso Unificado.* 2004.

KANEIWA, Ken, MIZOGUCHI, Riichiro y NGUYEN, Philip HP. 2015. *A Logical and Ontological Framework for Compositional Concepts of Objects and Properties.* s.l. : New Generation Computing, 2015.

Larman, C. 2008. *UML y patrones.* 2008.

Marin, Marvin David Arias. 2008. Catedra de Programación. *Catedra de Programación.* [En línea] 16 de octubre de 2008. [Citado el: 26 de octubre de 2015.] http://catedraprogramacion.forosactivos.net/t83-definicion-de-lenguaje-de-programacion-tipos-ejemplos.

MARTÍNEZ, Marilé Lemus, et al. 2016. *Sistema para la Informatización de la Gestión de Anuarios Estadísticos en la ONEI.* s.l. : Serie Científica-Universidad de las Ciencias Informáticas, 2016.

Martinez, Rafael. 2013. PostgreSQL-es. *PostgreSQL-es.* [En línea] 2013. [Citado el: 15 de diciembre de 2015.] http://www.postgresql.org.es/sobre_postgresql.

MDN. 2009. MDN. *MDN.* [En línea] 2009. [Citado el: 26 de octubre de 2015.] https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Introducci%C3%B3n.

MICHAUD, Frederic. 2015. *Identifying Key Attack Surface Resources with Dynamic Analysis.* 2015.

MORO, Alfonso Infante, PÉREZ, Oscar Gallego y MACÍAS, Amparo Sánchez. 2013. *Los gadgets en las plataformas de telefomación: el caso del proyecto DIPRO 2.0. Pixel-Bit: Revista de medios y educación, 2013.* 2013.

OBE, Regina O. y HSU, Leo S. 2015. *PostGIS in action.* s.l. : Manning Publications Co, 2015.

ÖVERGAARD, Gunnar y PALMKVIST, Karin,. 2008. *Use Cases: Patterns and Blueprints.* 2008.

Paradigm. 2010. Paradigm, 2010. *Paradigm, 2010.* [En línea] www.visual-paradigm.com, 2010. [Citado el: 15 de noviembre de 2015.] http://www.visual-paradigm.com/aboutus/newsreleases/vpuml80.jsp.

PAYER, Mathias, BARRESI, Antonio y GROSS, Thomas R. 2015. *Fine-grained control-flow integrity through binary hardening. En International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* s.l. : Springer International Publishing, 2015.

PHP. 2011. php. *php.* [En línea] php.net, 2011. [Citado el: 26 de octubre de 2015.] http://php.net/manual/es/preface.php.

Potencier, Fabien. 2015. librosweb. [En línea] 2015. [Citado el: 6 de Febrero de 2016.] http://librosweb.es/libro/symfony_1_1/capitulo_2/el_patron_mvc.html.

Potencier, François Zaninotto. 2009. Symfony, la guia definitiva. *Symfony, la guia definitiva.* [En línea] 2009. [Citado el: 20 de noviembre de 2015.] http://librosweb.es/libro/symfony_1_4/.

PRESSMAN, Roger. 2009. *Ingeniería del Software. Un enfoque práctico.* s.l. : McGraw-Hill/Interamericana, 2009.

prueba, Condiciones y casos de. Condiciones y casos de prueba. *Condiciones y casos de prueba.* [En línea] http://www.rmya.com.ar.

Reingart, Mariano. 2011. postgresql. *postgresql.* [En línea] 2011. [Citado el: 26 de octubre de 2015.] http://www.postgresql.org.ar/trac/wiki/PgAdmin?format=pdfarticle.

Sparx. 2000-2007. Sparx Systems. *Sparx Systems.* [En línea] Sparx Systems, 2000-2007. [Citado el: 21 de enero de 2016.] http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html. ISBN.

STEWART, Claire R., et al. 2016. *Sensory symptoms and processing of nonverbal auditory and visual stimuli in children with autism spectrum disorder.* s.l. : Journal of autism and developmental disorders, 2016.

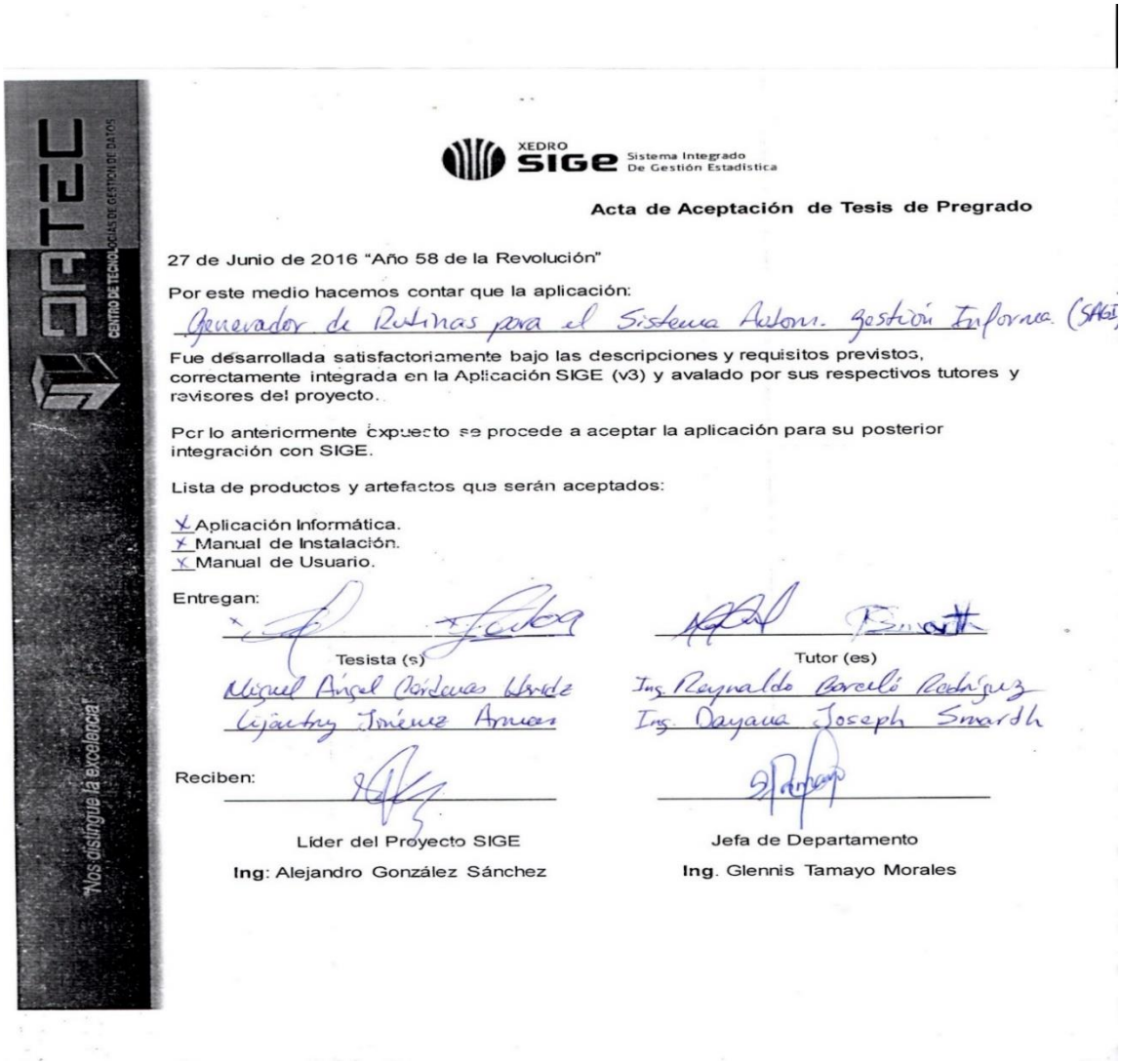
uml-diagrams. 2010. uml-diagrams. *uml-diagrams.* [En línea] 2010. [Citado el: 14 de octubre de 2015.] http://www.uml-diagrams.org/.

Anexos

Anexo 1. Preguntas realizadas al cliente como parte de la entrevista no estructurada realizada a los especialistas de La ONEI con el objetivo de recopilar información referente a la realización de rutinas mediante el SIGETools. Las mismas estaban enfocadas a:

1. Definición, uso y aspectos necesarios para la creación de un corte.
2. Definición, uso y aspectos necesarios para la creación de un nivel.
3. Definición, uso y aspectos necesarios para la creación de un modelo Virtual.
4. Criterios de búsqueda
5. Criterios de búsqueda para buscar niveles o modelos.

Anexo 2. Carta de Aceptación.



XEDRO SIGE Sistema Integrado De Gestión Estadística

Acta de Aceptación de Tesis de Pregrado

27 de Junio de 2016 "Año 58 de la Revolución"

Por este medio hacemos contar que la aplicación:

Generador de Rutinas para el Sistema Autom. Gestión Informa. (SAGI

Fue desarrollada satisfactoriamente bajo las descripciones y requisitos previstos, correctamente integrada en la Aplicación SIGE (v3) y avalado por sus respectivos tutores y revisores del proyecto.

Por lo anteriormente expuesto se procede a aceptar la aplicación para su posterior integración con SIGE.

Lista de productos y artefactos que serán aceptados:

X Aplicación Informática.
X Manual de Instalación.
X Manual de Usuario.

Entregan:

Tesista (s)
Miguel Ángel Córdovas Verde
Lisbeth Jiménez Arnaus

Tutor (es)
Ing. Reynaldo Borrell Rodríguez
Ing. Dayana Joseph Smardh

Reciben:

Líder del Proyecto SIGE
Ing: Alejandro González Sánchez

Jefa de Departamento
Ing. Glennis Tamayo Morales