Universidad de las Ciencias Informáticas

Facultad 6

Integración de Qooxdoo y Symfony 2

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: *Laritza López Lobaina*

Frank Yasel de Armas Mora

Tutores: *Frank Alberto Rodríguez solana*
Marcos Luis Ortíz Valmaseda

La Habana, junio de 2015

Año 57 de la Revolución

“Los que aseguran que es imposible, no deberían interrumpir a los que estamos intentando”

Tomás Edison

DECLARACIÓN DE AUTORÍA

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

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

Laritza López Lobaina

_____
Firma del Autor

Marcos Luis Ortíz Valmaseda

_____
Firma del Tutor

Frank Yasel de Armas Mora

_____
Firma del Autor

Frank Alberto Rodríguez Solana

_____
Firma del Tutor

DATOS DE CONTACTO

Ing. Marcos Luis Ortíz Valmaseda:
Especialidad de graduación: Ingeniero en Ciencias Informáticas
Categoría docente: Especialista "B" en Ciencias Informáticas
Categoría científica: Ninguna
Años de graduado:
Email: mlortiz@uci.cu

Ing. Frank Alberto Rodríguez Solana:
Especialidad de graduación: Ingeniero en Ciencias Informáticas
Categoría docente: Especialista "B" en Ciencias Informáticas
Categoría científica: Ninguna
Años de graduado:
Email: frankalberto@uci.cu.

AGRADECIMIENTOS

Laritza:

Agradezco a mis padres, pues ellos junto a mi han realizado los mayores sacrificios, sin su ayuda nada de esto hubiese sido posible, no existen palabras para agradecerles, ni para decirles lo mucho que los quiero.

A mi familia por todo el cariño y la comprensión demostrada todo este tiempo, en especial a mi tía por acogerme como una hija.

A mi novio Carlos por apoyarme en todo este tiempo que hemos estado juntos, por ser mi leal compañero en los momentos más difíciles y por ser la solución en muchos de estos momentos, por quererme y comprenderme, por todos los momentos felices que hemos pasado juntos.

.

AGRADECIMIENTOS

Frank:

Agradecer a todas las personas que de una manera u otra estuvieron durante mi vida en estos 5 años. Primero que todo agradecer a mi mamá, a mi papá y a mis abuelos que por razones y cosas de la vida no pueden estar presentes. Agradecer a todos mis amigos que me han soportado desde 1er año Bidot, Eric, Jose Eduardo, Reidel, Rolando, Yeny y Arianna más todas la gente del grupo 6404.Agradecer a Andy con sus casas de estudio y Alejandro sin entender. A Nurisely, Yadira a Yuya el Jeem y las muchachitas del 104 Yai y Ana. Agradecimientos muy especiales a Danaysis por ser amiga y mujer. Agradecer también a los muchachos del aula nueva con los que hice muy buenas relaciones y también amigos: a Alain y Sonia que fueron durante estos últimos años como nuestros padres a Markitos por ser así y no cambiar gracias a eso pudimos divertirnos con él, a Yoan por ser no solo compañero de cuarto sino amigo también a Arian el Sury que fue el que nos vivió todo este tiempo, al Yainel y Alex con sus vicios de café y también Daldis con el mejor café de todos los tiempos y por supuesto como olvidar a la gente del domino. Dar también agradecimientos especiales a nuestro trío Carlos que nos guió cuando más lo necesitábamos. A mi dúo de tesis Bichiti que no tengo palabras para esa niña, me hizo caminar por toda la UCI con una mochila rosada con estrellitas (de macho alfa), sin comentarios. A Sheyla que pese a no estar aquí en la universidad me apoyó en todo lo que pudo. A mis amigos de la infancia Raude, Josue y Yojan que me llevan cargando y yo a ellos desde que nacimos. Agradecer a mi amiga Arleidis que no pudo estar presente este día, sobre todo agradezco a todos los que se encuentran aquí y muchas personas que no mencioné porque la lista es grande y si sigo no tengo donde parar a todos ellos también los quiero.

DEDICATORIA

Laritza:

Le dedico esta tesis a mi hermano, a mi mama, mi papa y mis abuelos.

Frank:

Le dedico esta tesis a mis padres y mis abuelos que me encaminaron en la vida y hoy soy lo que soy es gracias a ellos.

Resumen

La Universidad de las Ciencias Informáticas es una institución que dentro de sus funciones está el desarrollo de *software*. En el Centro de Tecnologías de Gestión de Datos (DATEC) la mayoría de los proyectos productivos están desarrollados utilizado los *frameworks ExtJS* y *Symfony 2*. A partir de la versión 2.1 de *ExtJS* su licencia es privativa para los productos comercializables, por lo que se decide utilizar *Qooxdoo* 4.1 como alternativa al ser este de código abierto. Debido a que no existe ningún *bundle* que permita integrar los *frameworks Qooxdoo* y *Symfony 2* la utilización de estas 2 herramientas para el desarrollo de aplicaciones web se ve limitada. El presente trabajo de diploma está orientado al desarrollo de un *bundle* que integre los *framework Qooxdoo* y *Symfony 2* el cual facilitará el desarrollo de aplicaciones web utilizando las potencialidades de estas herramientas. Para guiar el proceso se utilizó la metodología de desarrollo de *software OpenUP*. Para el modelado de la solución se utilizó Visual Paradigm v8.0, como Entorno Integrado de Desarrollo (IDE) PHPStorm v8.0 y como servidor web Apache HTTP Server v2.2. Para la implementación se utilizó como lenguaje de programación del lado del cliente *JavaScript* con el *framework Qooxdoo* 4.1 y como lenguaje del lado del servidor PHP con el *framework Symfony 2*. Como resultado se obtendrá un *bundle* que permite la integración de *Qooxdoo 4.1* y *Symfony 2*.

Palabras Claves: Integración, *Qooxdoo, Symfony 2, Bundle*.

Abstract

The University of Informatics Sciences is an institution that has among its functions to develop software. In the Data Management Technologies Center (DATEC) most of the productive projects are developed using the frameworks ExtJS and Symfony 2. From version 2.1 of ExtJS its license is private for trading products, so it is decided to use Qooxdoo 4.1 as an alternative for being this open source. Because there is no bundle that allows integrating Qooxdoo and Symfony 2 frameworks, using these two tools for developing web applications is limited. This work is aimed at developing a bundle that integrates Qooxdoo and Symfony 2 framework which will facilitate the development of web applications using the potential of these tools. To guide the process the software development methodology OpenUP was used. For modeling the solution it was used Visual Paradigm v8.0, as Integrated Development Environment (IDE) v8.0, and as web Apache server HTTP Server v2.2. To implement it was used as the programming language by the client JavaScript with Qooxdoo 4.1 framework, and as language by the server PHP with Symfony framework 2. The result is a bundle that allows the integration of Qooxdoo 4.1 and Symfony 2.

Keywords: Integration, Qooxdoo, Symfony 2, Bundle.

ÍNDICE

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) se encargan de estudiar, desarrollar, implementar, almacenar y distribuir la información utilizando *hardware* y *software* como medios informáticos. El adelanto tecnológico combinado con el auge de las ciencias informáticas, ha propiciado que la complejidad de los proyectos de desarrollo de *software* se haya incrementado de manera considerable. Debido al surgimiento de nuevas tecnologías y lenguajes de programación ha aumentado la complejidad en el desarrollo de aplicaciones web, con el objetivo de agilizar, organizar y automatizar estas tareas surgen los *frameworks* de desarrollo.

Un *framework* o marco de trabajo es una estructura conceptual y tecnológica de soporte definida, con artefactos o módulos de *software* concretos, en base a la cual otro proyecto de *software* puede ser organizado y desarrollado. (Zaninotto 2008) Dentro de los *frameworks* de desarrollo de aplicaciones web se encuentra *Symfony 2*, este es un completo marco de trabajo para el desarrollo de aplicaciones web basado en el patrón Modelo-Vista-Controlador. Es una colección de más de veinte bibliotecas independientes que se pueden utilizar dentro de cualquier proyecto PHP y contienen lo necesario para casi cualquier situación, independientemente de cómo se desarrolla el proyecto. (Boyarynov 2011) En *Symfony 2* todo está conformado por *bundles.* Un *bundle* es un concepto similar al de los *plugins*[1] en otras aplicaciones. Estos son la parte más importante de *Symfony 2*, permiten utilizar funcionalidades construidas por terceros o empaquetar tus propias funcionalidades para distribuirlas y reutilizarlas en otros proyectos. Además, facilitan la activación o desactivación de determinadas características dentro de una aplicación.

Para la presentación de las aplicaciones, tanto web dinámicas como *Single Page Application* (SPA)[2] se utilizan varios *frameworks JavaScript* para el desarrollo de la parte del cliente, entre las cuales se pueden nombrar *JQuery, ExtJS y Qooxdoo. Qooxdoo*, es un framework *JavaScript*, desarrollado con el objetivo de crear aplicaciones para una amplia gama de plataformas. Con su modelo de programación orientada a objetos permite crear aplicaciones ricas e interactivas (RIAs), aplicaciones nativas para los dispositivos móviles, las aplicaciones web tradicionales o incluso aplicaciones para correr fuera del navegador.(Developed 2014)

[1] Aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. Puede nombrarse al *plugin* como un complemento.

[2] Aplicación de una sola página (SPA), también conocida como interfaz de una sola página (SPI), es un sitio web o aplicación web que cabe en una sola página web con el objetivo de proporcionar una experiencia de usuario más fluida similar a una aplicación de escritorio.

INTRODUCCIÓN

Qooxdoo trata de aprovechar las características de JavaScript, se basa en gran medida en los *namespaces* para permitir una fácil integración con otras bibliotecas y código de usuario existente. La mayoría de los navegadores modernos son compatibles (por ejemplo, Firefox, Internet Explorer, Opera, Safari, Chrome). Viene con una API (*Aplication Programming Interface*) de referencia muy completa. El rápido y completo parser no sólo permite la generación de documentación, sino que es una parte integral del proceso de construcción automático que hace la optimización, la compresión, la vinculación y el despliegue de aplicaciones personalizadas muy fácil de usar.
En la Universidad de las Ciencias Informáticas, específicamente en el centro DATEC, los productos realizados en su mayoría se encuentran desarrollados con el framework *ExtJS*. A partir de la versión 2.1 del framework *ExtJS* la licencia es *General Public License* (GPL v3), la cual le permite a los desarrolladores usar esta herramienta pero deben liberar código y en caso de usarla con fines comerciales deben comprar la licencia comercial. Por lo que surge la necesidad de utilizar *Qooxdoo* integrado a *Symfony 2,* debido a que en la UCI se está llevando a cabo el proceso de independencia tecnológica y además comprar el derecho de licencia por cada proyecto que se desarrolla sería algo costoso para la universidad.(Sencha 2015) Al mismo tiempo se quiere hacer uso de las ventajas que provee *Qooxdoo* el cual es un nuevo *framework* con funcionalidades específicas encaminadas a mejorar el desarrollo de aplicaciones web. El trabajo con estas dos herramientas permitiría desarrollar tanto aplicaciones web dinámicas como SPA, por lo que se plantea como **problema a resolver** ¿cómo lograr la integración de *Qooxdoo y Symfony 2*?

A partir del problema planteado se define como **objeto de estudio** creación de *bundle*s para *Symfony 2* y como **campo de acción** creación de *bundle* de *Qooxdoo para Symfony 2.*

Para darle solución al problema se define como **objetivo general** desarrollar un *bundle* que permita la integración de los *frameworks Qooxdoo* y *Symfony 2*, para la realización del objetivo general se plantearon los siguientes **objetivos específicos**:

- Realizar el marco teórico de la investigación para la integración de *Qooxdoo y Symfony 2.*
- Fundamentar las herramientas a utilizar para realizar la integración de *Qooxdoo y Symfony 2.*
- Realizar el análisis y diseño para el desarrollo del *bundle Qooxdoo y Symfony 2.*
- Implementar el *bundle* para integración de *Qooxdoo y Symfony 2.*
- Realizar las pruebas para validar la correcta ejecución del sistema.

Para cumplir con los objetivos planteados se trazaron las siguientes **tareas de la investigación**:

- Análisis de los principales conceptos asociados al desarrollo de *bundles* de *Symfony 2.*
- Análisis de la estructura y el diseño de los *bundles* existentes para integrar las bibliotecas *JavaScript* a *Symfony 2*.
- Definición de las herramientas informáticas y metodologías a usar en el desarrollo del *bundle* para la integración de los *framework Qooxdoo* y *Symfony 2.*
- Realización del análisis de las funcionalidades con la que debe contar el *bundle Qooxdoo-Symfony 2* para generar la especificación de las características operacionales del mismo.
- Elaboración de los artefactos y modelos asociados a la metodología de desarrollo de *software* escogida para guiar el proceso de implementación.
- Implementación de los principios de diseño y funcionamiento para satisfacer las necesidades funcionales y tecnológicas del *bundle Qooxdoo-Symfony 2.*
- Validación del sistema a partir de las pruebas de *software* para comprobar su correcto funcionamiento.

Preguntas Científicas.

- ¿Cuáles son los fundamentos teóricos que proporcionan la base para el desarrollo del *bundle Qooxdoo-Symfony 2*?
- ¿Cuáles son las características y capacidades que debe tener el *bundle Qooxdoo-Symfony 2* para permitir la integración de estas tecnologías?
- ¿Cómo se debe estructurar el proceso de desarrollo del *bundle Qooxdoo-Symfony 2* para lograr la realización de aplicaciones web utilizando estas tecnologías?
- ¿Cómo validar el correcto funcionamiento del *bundle Qooxdoo-Symfony 2*?

Para dar solución al problema correspondiéndose con el objetivo general trazado se aplicaron los siguientes métodos de la investigación científica:

Métodos teóricos:

- **Analítico-Sintético:** se consultó la bibliografía necesaria para dar cumplimiento a la fundamentación del estado del arte y la consulta de la literatura especializada de los *frameworks*

Symfony 2 y *Qooxdoo*. A partir de este análisis se logró obtener toda la información requerida para sintetizar el marco teórico de la presente investigación.

- **Análisis Histórico-Lógico:** utilizado para explorar los historiales de herramientas y tecnologías, utilizadas para la implementación de *bundle*s que integren *framework JavaScript* a *Symfony 2*.

Métodos empíricos:

- **Observación:** este método se usó para realizar valoraciones y obtener informaciones a partir de lo observado. Esto se manifiesta principalmente cuando se realizan observaciones sobre el funcionamiento de *bundle*s similares al desarrollado, lo que da una visión de cómo tiene que ser el sistema a realizar en su forma externa.

El trabajo está estructurado en 3 capítulos.

- En el primer capítulo se concibe el análisis teórico de la investigación, aquí se definen las herramientas a utilizar en la implementación, así como la metodología de desarrollo de *software*, las tecnologías a emplear y sus fundamentos en la selección para el proceso de desarrollo.
- En el segundo capítulo se representan todas las características que poseerá el futuro sistema. Se realizará el levantamiento de los requisitos funcionales y no funcionales, y a partir de ellos los casos de uso del sistema con sus respectivas descripciones textuales.
- En el tercer capítulo se muestra el modelo de implementación como resultado del diseño anteriormente desarrollado. Como parte del mismo se presenta el diagrama de componentes y los estándares de codificación utilizados para el desarrollo de la aplicación. Además se describen las pruebas a realizar, con el objetivo de comprobar el correcto funcionamiento del *bundle* Qoooxdoo-*Symfony* 2 en distintos momentos del ciclo de vida del *software*.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se analizarán temas relacionados con los diferentes *bundle*s que integran los diversos *frameworks JavaScript* a *Symfony 2*. Este análisis tiene como objetivo definir las herramientas de acuerdo con las características más factibles a utilizar para el desarrollo. Se brinda una descripción de la metodología a emplear y el fundamento de su selección. También se exponen las principales herramientas, lenguajes y tecnologías que se utilizarán en el desarrollo de la integración.

En la investigación que se realiza aparecen una serie de conceptos que se encuentran asociados al desarrollo del problema. Para lograr un mayor entendimiento por parte del lector y una mayor visión de los temas tratados, a continuación se puntualizan cada uno de ellos.

1.1. Integración se software

Una de las fases del ciclo de vida del *software* es la de integración. Es imprescindible poder integrar los desarrollos de *software* en forma de productos y soluciones para que puedan ponerse en uso. Esto exige manejarse en varias disciplinas, tener en cuenta el punto de vista del usuario, definición y aplicación de procedimientos con rigor, llevar a cabo gestiones de configuración, etc. Algunas tareas que suelen realizarse con la integración del *software* es la creación de paquetes, integración de servicios para entornos de trabajo en grupo y migraciones de sistemas.(Olea 2014)

1.2. Bundle

Los *bundles* son la base de la nueva filosofía de trabajo de *Symfony 2*. El código de las aplicaciones y el propio código fuente de *Symfony 2* se estructura mediante *bundles*. Técnicamente, un *bundle* es un directorio que contiene todo tipo de archivos dentro una estructura jerarquizada de directorios. (Eguiluz 2014)

Los *bundles* de las aplicaciones de *Symfony 2* suelen contener clases PHP y archivos web (*JavaScript*, *CSS* e imágenes). No obstante, no existe ninguna restricción sobre lo que puedes incluir dentro de un *bundle*. Tampoco existen límites técnicos sobre el tamaño que puede llegar a tener un *bundle*. (Eguiluz 2014)

Algunos programadores prefieren almacenar todo el código de la aplicación en un único *bundle* gigantesco. Lo hacen porque así creen que es más fácil reutilizar el *bundle* en otros proyectos. Otros programadores prefieren usar tantos *bundles* como *divisiones lógicas* tenga la aplicación. (Eguiluz 2014)

Los *bundles* presentan una estructura de directorios simple y flexible. Por defecto el sistema de *bundles*

sigue una serie de convenciones que ayudan a mantener el código consistente entre todos los *bundles* de *Symfony 2*. A continuación se describe cómo van a estar compuesto los *bundles*:

- Controller/: contiene los controladores del *bundle*, a través de los cuales se procesan las peticiones de la vista y luego se devuelven las respuestas.
- DependencyInjection/: contiene las extensiones para las clases de inyección de dependencias, la configuración que importan los servicios y registra uno o más pases del compilador (este directorio no es obligatorio).
- Resources/config/: contiene los archivos de configuración, incluyendo la configuración de enrutamiento (por ejemplo, routing.yml).
- Resources/views/: contiene las plantillas organizadas según el nombre del controlador (por ejemplo, Hello/index.html.twig).
- Resources/public/: contiene recursos web (imágenes, hojas de estilo, etc.) y es copiado o enlazado simbólicamente al directorio web/ del proyecto con el comando assets: install.
- Tests/: tiene los tests unitarios y funcionales del *bundle*. (Weaver 2014)

Actualmente existen diversos *bundles* que integran los *frameworks JavaScript* más conocidos a *Symfony 2*, a continuación se hará referencia a algunos de estos.

JQuery

Tabla 1- Bundles de jQuery para Symfony 2

Nombre del Proyecto	Creador	Año de Creación	URL
sonata-project/jquery-*bundle*	Thomas Rabaix , Sonata Community	2010	https://github.com/sonata-project/SonatajQuery*Bundle*
tubssp/*Symfony* -jquery	Andreas Lemke	2007	https://github.com/tubssp/*Symfony* -jquery
Symfony -*bundle*/jquery-*bundle*	Hassan Amouhzi	2014	https://github.com/*Symfony* -*bundle*/jquery-*bundle*

rhapsody-project/jquery-*bundle*	rhapsody-project	2014	https://github.com/rhapsody-project/jquery-*bundle*

En el caso de los bundle del framework jQuery, para utilizarlos solo es necesario incluir los archivos que contiene a nuestro proyecto y añadir los mismos a la página base de la siguiente manera:

Fig. 1- Ejemplo de añadir biblioteca jQuery a un proyecto Symfony2

La figura 1 ilustra las líneas de código necesarias para incluir la biblioteca *jQuery* a la aplicación, la misma fue tomada del archivo de configuración del *bundle SonatajQueryBundle*. Como se puede observar para poder utilizar *jQuery* en los proyectos de *Symfony 2* solo sería necesario copiar dicha biblioteca para el proyecto e incluirla.

Algunos *bundles* como el *jquery-bundle* de *Rhapsody* presentan comandos para instalar los assets[3] correspondientes a *jQuery*, los mismos son de gran ayuda para el usuario, ya que les evita el trabajo de tener que copiar manualmente estos archivos.

ExtJS

Tabla 2- Bundles de ExtJS para Symfony 2

Nombre del Proyecto	Creador	Año de Creación	URL
jamesmoey/extjs-*bundle*	James Moey	2013	https://github.com/jamesmoey/extjs-*bundle*

[3] Los *assets web* son las hojas de estilo CSS, los archivos JavaScript y las imágenes que se utilizan en el *frontend* de las aplicaciones para que tengan un buen aspecto.

AlexOrphan/*Symfony* 2-extjs4	Alex Orphan	2014	https://github.com/AlexOrphan/*Symfony* 2-extjs4

En el caso del framework *ExtJS* la creación de los *bundles* se comporta de manera diferente, dichos *bundles* presentan una estructura similar a la siguiente:

Annotation	Fix bug option, reader and writer parameters was not being apply if v...	2 years ago
Command	fix of big problem in GenerateRestControllerCommand::getSkeletonDirs	2 years ago
Component	fixed a warning i get	2 years ago
Controller	add phpdoc to $file var in GeneratorController::generateModelAction	2 years ago
DependencyInjection	Fix failed Continue Integration. use hasDefinition to check existence...	2 years ago
Generator	Move directory structure to be in line with other standard symfony bu...	2 years ago
Listener	Check property reflection before using it.	a year ago
Resources	add compiler pass to to replace naming_strategy argument in nelmio_ap...	2 years ago
Service	Try to get association key for OneToOne relationship from the inverse...	a year ago
Tests	fix callable expected	a year ago
Twig	Issue #11	2 years ago

Fig. 2- Estructura del bundle extjs-bundle

En la figura 2 se muestra la estructura del *bundle extjs-bundle* desarrollado por James Moey reconocido contribuidor de la comunidad de *Symfony 2*. Además de este *bundle* existen otros mencionados anteriormente los cuales presentan características similares como son:

- Comandos para crear un *bundle* que contenga la biblioteca *ExtJS*.
- Comandos para generar los controladores para *ExtJS*.
- Comandos para generar el CRUD de una entidad, siguiendo la estructura de *ExtJS*.

Después de analizado cómo se comporta el desarrollo de los *bundles* por la comunidad de *Symfony 2* para las bibliotecas *jQuery* y *ExtJS*, se decide que para la realización de un *bundle* que integre *Qooxdoo* a *Symfony 2* es necesario que este cuente con un conjunto de funcionalidades que le facilite al usuario final el desarrollo de aplicaciones utilizando estas tecnologías. Dichas funcionalidades deben permitir:

- Crear *bundles* con la estructura de *Qooxdoo*.
- Compilar *bundles* de *Qooxdoo*.

- Generar el CRUD de las entidades de los *bundles* de *Qooxdoo*.

1.3. Metodología de desarrollo de software

Cuando se inicia el desarrollo de un *software* se debe tener en cuenta qué metodología escoger para guiar el ciclo de vida del *software*. Las metodologías de desarrollo consisten en un conjunto de procedimientos, técnicas, herramientas y soporte documental que ayuda a los desarrolladores a realizar un nuevo *software*. Durante su ciclo de vida indica qué es lo que hay que obtener durante todo el proceso de desarrollo, es decir, cómo se obtienen los productos parciales y finales pero no cómo hacerlos.(Santos 2010)

En la actualidad se cuenta con metodologías tradicionales o robustas y metodologías ágiles. Las tradicionales están pensadas para el uso exhaustivo de documentación durante todo el ciclo vida del proyecto. En cambio las metodologías ágiles ponen vital importancia a la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y en mantener una buena relación con el cliente.

El éxito del producto depende en gran parte de la metodología escogida por el equipo, ya sea tradicional o ágil, donde los equipos maximicen su potencial y aumenten la calidad del producto con los recursos y tiempos establecidos.(Cabrera 2008)

A continuación se muestra una comparación entre las metodologías ágiles y tradicionales. (Penadés 2011)

Tabla 3- Comparación entre metodologías ágiles y tradicionales

Metodologías Ágiles	Metodologías Tradicionales
Especialmente preparados para cambios durante el proyecto	**Cierta resistencia a los cambios**
Proceso menos controlado, con pocos principios	**Proceso mucho más controlado, con numerosas políticas o normas**
No existe contrato tradicional o al menos es bastante flexible	**Existe un contrato prefijado**
El cliente es parte del equipo de desarrollo	**El cliente interactúa con el equipo de desarrollo mediante reuniones**
Grupos pequeños y trabajando en el mismo sitio	**Grupos grandes y posiblemente distribuidos**
Pocos artefactos	**Más artefactos**

Pocos roles	Más roles
Menos énfasis en la arquitectura del software	**La arquitectura del software es esencial y se expresa mediante modelos**

Analizadas las características anteriores de cada tipo de metodologías y teniendo en cuenta las condiciones con las que se necesita implementar el *software*, se decide emplear una metodología ágil para guiar el proceso de desarrollo de *software* del *bundle* que integra los *frameworks Qooxdoo y Symfony 2.*

1.3.1. Metodología ágil XP

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Es conveniente escoger esta metodología para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

A continuación se presentarán las características esenciales de XP organizadas en las historias de usuario, proceso y ciclo de vida.(Wesley 2000)

Historias de Usuario:

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.(Jeffries 2001)

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

- 1- El cliente define el valor de negocio a implementar.
- 2- El programador estima el esfuerzo necesario para su implementación.
- 3- El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.

- 4- El programador construye ese valor de negocio.
- 5- Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el *software* o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.(Jeffries 2001)

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Fase I: Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Fase II: Planificación de la Entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

Fase III: Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas.

Fase IV: Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Fase V: Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de

tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

Fase VI: Muerte del Proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.(Jeffries 2001)

1.3.2. Metodología ágil Scrum

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto. Éstas son las verdaderas protagonistas, especialmente la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

En el ciclo de vida de *Scrum* cada funcionalidad es escrita en una carta, cada carta es priorizada por el cliente y estimada por el equipo de desarrollo. Luego cada carta es asignada a una iteración. Cuando se implemente, se muestra al cliente y se da como aceptada en caso de satisfacer al mismo.(Schwaber 2001)

Prácticas de *Scrum*:

1-**Product Backlog:** Es donde se define todo lo que se necesita en el producto final, basado en el conocimiento actual. Se compone de una lista de prioridades que constantemente son actualizadas según las necesidades operativas y técnicas para que el sistema esté siendo construido o mejorado.

2-**Effort estimation:** Es un proceso iterativo, en el que las estimaciones de los elementos de la reserva del producto se centran en un nivel más preciso debido a una mayor cantidad de información disponible sobre un tema determinado. El dueño del producto, junto con el Equipo Scrum son los responsables de realizar la estimación del esfuerzo.

3-**Sprint:** *Sprint* es el procedimiento de adaptación a las necesidades cambiantes de las variables ambientales, tiempo, recursos, conocimientos, tecnología, etc.) El Equipo *Scrum* se organiza para producir

nuevos ejecutables mediante *Sprints* de aproximadamente treinta días naturales de duración. Las herramientas de trabajo del equipo son las reuniones de planificación de *Sprint*, la Pila de *Sprint* y reuniones diarias de *Scrum*.

4-**Sprint Planning meeting:** Es una reunión de dos fases, organizada por el *Scrum Master*. Los clientes, usuarios, administradores, dueño del producto y el Equipo *Scrum* participar en la primera fase de la reunión para decidir sobre los objetivos y las funcionalidades del siguiente *Sprint*. La segunda fase de la reunión se lleva a cabo por el *Scrum Master* y el Equipo *Scrum* se centra en cómo el incremento del producto se lleva a cabo durante el *Sprint*.

5-**Sprint Backlog:** Es el punto de partida de cada *Sprint*. Se trata de una lista de elementos seleccionados del Product Backlog para ser implementados en el próximo *Sprint*

6-**Daily Scrum meeting:** Reuniones diarias de *Scrum* se organizan para realizar un seguimiento de los progresos del Equipo *Scrum* de forma continua y también sirven como reuniones de planificación.

7-**Sprint Review meeting:** En el último día de cada Sprint, el Equipo *Scrum* y el *Scrum* Master presentan los resultados del *Sprint*. Los participantes evalúan el incremento del producto. La reunión de revisión puede provocar pedidos nuevos e incluso cambiar la dirección del sistema que se está desarrollando.(Schwaber 2001)

1.3.3. Metodología ágil OpenUP

OpenUp, es una metodología de desarrollo de *software* basado en *Rational Unified Process* (*RUP*), que mantiene las mismas características pues está dirigido por casos de uso, centrado en la arquitectura y además es iterativo e incremental. Es un proceso interactivo de desarrollo de *software* simplificado, completo y extensible.

Está diseñado para soportar equipos pequeños, no dispersos y que trabajan en proyectos cuya duración está entre tres y seis meses, debido a que permite disminuir las probabilidades de fracaso en los proyectos e incrementar las probabilidades de éxito. Permite detectar errores en fases tempranas a través de un ciclo iterativo. Una de sus principales ventajas está dada por la poca elaboración de documentación, diagramas e iteraciones requeridos en la metodología RUP. Dicha metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

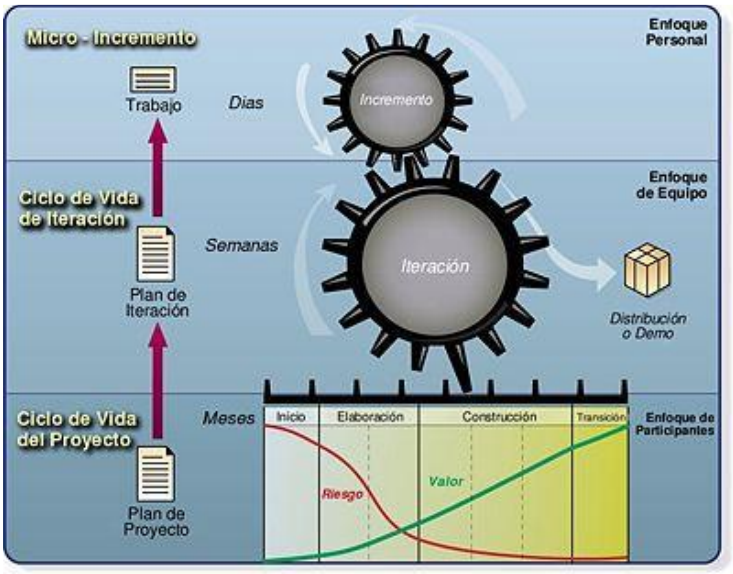

Fig. 3- Ciclo de vida de OpenUP

OpenUP es un proceso iterativo cuyas iteraciones se distribuyen a través de cuatro fases: Concepción, Elaboración, Construcción y Transición.

Fases de la metodología *OpenUP*:

- **Concepción (Inicio):** es la primera de las 4 fases en el proyecto del ciclo de vida, acerca del entendimiento del propósito y objetivos, obteniendo suficiente información para confirmar qué el proyecto debe hacer. El objetivo de esta fase es capturar las necesidades de los *stakeholder* , en los objetivos del ciclo de vida para el proyecto.
- **Elaboración:** es la segunda de las 4 fases del ciclo de vida del *OpenUP* donde se trata los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base y la elaboración de la arquitectura del sistema.
- **Construcción:** esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.
- **Transición:** es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y performance del último entregable de la fase de construcción. Lenguajes de programación del lado del cliente y del lado del servidor. (Eclipse 2012)

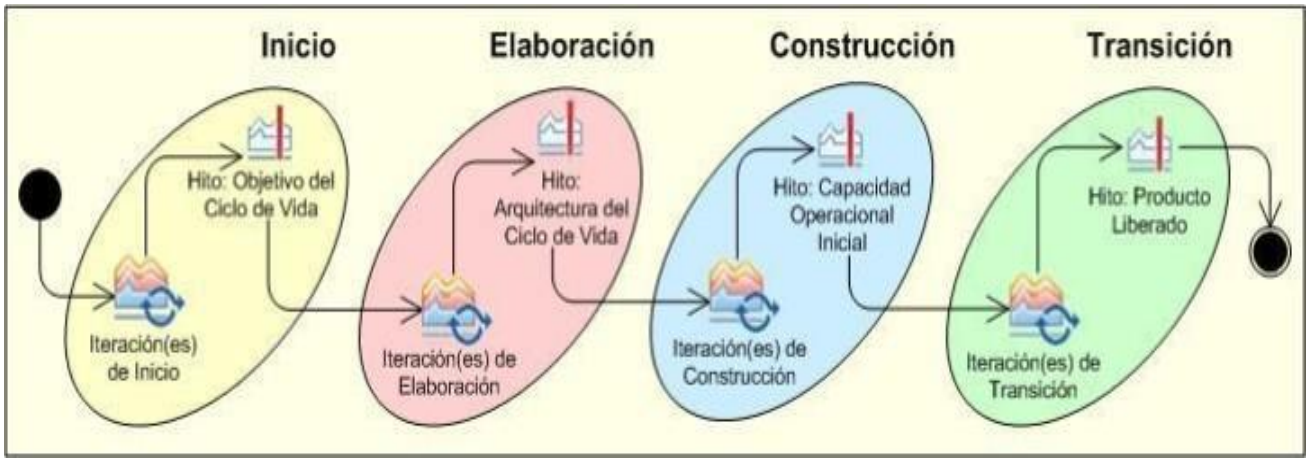

Fig. 4-Iteraciones de OpenUP

Luego de haber estudiado las características fundamentales de algunas metodologías ágiles de *software*, se seleccionó *OpenUP* para guiar el proceso de desarrollo del *bundle* que integra los *frameworks Qooxdoo* y *Symfony 2*, ya que se adapta al ambiente del proyecto por ser una metodología ágil además es perfecta para grupos de trabajo pequeños, tiempo de trabajo limitado y genera una cantidad mínima de artefactos.

1.4. Lenguaje de modelado

Para establecer una coordinada comunicación entre los miembros de un equipo de desarrollo de *software* el lenguaje de modelado constituye una forma común. Este lenguaje sirve de herramienta para desarrollar una representación simplificada de la realidad a través de abstracciones que se plasman en notaciones gráficas y sirven de apoyo al análisis de problemas representándolo en forma más intuitiva para personas sin especialización en Informática.

1.4.1. Lenguaje Unificado de Modelado (UML 2.0)

UML (por sus siglas en inglés Unified Modeling Language) es un lenguaje de modelado visual de propósito general orientado a objetos. Cubre las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de *software*.

Permite la representación conceptual y física de un sistema. UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de lo que se quiere representar. Es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad. UML ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales tales como procesos de negocios y funciones del

sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de *software* reutilizables.

UML es un lenguaje de modelado visual que sirve para: visualizar, especificar, construir, documentar sistemas independientemente de la metodología de análisis y diseño pero siempre con una perspectiva orientada a objetos.(Ruiz 2010)

Ventajas

- Permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Permite especificar cuáles son las características de un sistema antes de su construcción.
- A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.
- Se basa en una notación gráfica concisa, fácil de aprender y utilizar.
- Se puede utilizar para modelar sistemas en diversos dominios: sistemas de informaciones empresariales, sistemas Web, sistemas críticos y de tiempo real, incluso en sistemas que no son software.(Ruiz 2010)

1.4.2. Visual Paradigm para UML 8.0

Para el desarrollo del modelado de datos se utilizará *Visual Paradigm* 8.0, esta es una herramienta CASE, desarrollada por la compañía *Visual Paradigm International*, que utiliza UML 2.0 como lenguaje de modelado. Esta soporta el ciclo de vida completo del desarrollo de software: Análisis y Diseño orientados a objetos, Construcción, Pruebas y Despliegue.

Visual Paradigm 8.0 posibilita la generación de código desde diagramas y la generación de documentos en varios formatos como HTML, *Microsoft Word* y PDF. De igual forma facilita la generación de código y la ingeniería inversa en lenguajes como *Java*, C++, CORBA IDL, PHP, XML *Schema, Ada y Python*. (Paradigm 2012)

1.5. Lenguaje de programación

1.5.1. JavaScript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas que permitan intercambiar con los usuarios. Se implementa como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Es interpretado, por lo que no es

necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con *JavaScript* se pueden probar directamente en cualquier navegador que lo soporte, sin necesidad de procesos intermedios. (Eguiluz 2007)

1.5.2. PHP 5.3

La elección del lenguaje de programación se sustentó en las particularidades de la solución a desarrollar. Como el *framework* de desarrollo seleccionado es *Symfony 2 en su* versión *2.3.7* para su uso se requiere PHP 5.3.3 o una versión superior.

PHP (**PHP** *Hypertext Preprocessor*) es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor (*server - side scripting*) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.

Entre las principales características están:

- Es un lenguaje multiplataforma.
- Completamente orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una Base de Datos.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con *MySQL y PostgreSQL*.
- Posee una amplia documentación en su página oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. (Torres 2008)

1.6. *Framework* o Marco de Trabajo

Los frameworks son soluciones completas que contemplan herramientas de apoyo a la construcción (ambiente de trabajo o desarrollo) y motores de ejecución (ambiente de ejecución). Tienen como objetivo brindarles a los desarrolladores una mejor organización y estructura de sus proyectos proporcionando una arquitectura definida la cual ayuda hacer sus aplicaciones con mayor rigidez.(SOA 2010)

Un *framework* o marco de trabajo es un conjunto de procesos, sistemas, conceptos y tecnologías, que brindan al desarrollador la posibilidad de reutilización de componentes y está especializado en facilitar el desarrollo del *software*.

1.6.1. Qooxdoo

CAPÍTULO-I: FUNDAMENTACIÓN TEÓRICA

Qooxdoo es un *framework* o marco de trabajo *JavaScript*, que permite crear aplicaciones para una amplia gama de plataformas. Con su modelo de programación orientada a objetos se pueden construir aplicaciones ricas e interactivas, aplicaciones nativas para dispositivos móviles, aplicaciones web tradicionales o incluso aplicaciones de escritorio. Qooxdoo posee código abierto bajo licencias liberales. (Developed 2014)

Características principales:

Tiempo de ejecución:

Qooxdoo compatible con una amplia gama de entornos de JavaScript:

- Navegadores de escritorio (Internet Explorer, Firefox, Opera, Safari, Chrome)
- Navegadores móviles (iOS, Android, Windows Phone)
- Navegador menos motores JS (Node.js, Rhino)
- No requiere plugins
- Modificaciones no críticas de los objetos nativos de JavaScript para permitir una fácil integración con otras bibliotecas y código personalizado.

La orientación a objetos:

- Marco se basa totalmente en clases.
- Además de las clases regulares, ofrece clase abstracta, estática o singleton.
- Constructores y destructores.
- Miembros públicos, protegidos y privados por convención de nombres, que pueden (en parte) se cumplan durante el desarrollo.

Programación:

- Puro JavaScript
- No requiere conocimientos de HTML
- No se requieren conocimientos de CSS
- No se requieren conocimientos de DOM
- Soporte completo para la programación basada en eventos
- Diseñado para un alto rendimiento
- Muchas aplicaciones de ejemplo

Internacionalización:

- Construido en la internacionalización (i18n) y localización de apoyo (l10n)

- Apoyo a todos los idiomas y configuraciones regionales, al menos de este planeta
- Con base en los datos del repositorio integral Locale Unicode Común (CLDR)
- Formato de archivo de traducción conocida (.po)

Referencia de la API

- La funcionalidad de búsqueda
- Ayuda de la API Online y Offline.

1.6.2. Symfony 2.3.7

Symfony 2 ha sido ideado para aprovechar al límite todas las nuevas características de PHP 5 y por eso es uno de los *frameworks* PHP con mejor rendimiento. Su arquitectura interna está completamente desacoplada, lo que permite reemplazar o eliminar fácilmente aquellas componentes que no encajan en tu proyecto. Es un completo marco de trabajo diseñado para optimizar el desarrollo de las aplicaciones web utilizando los componentes que proporciona. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web utilizando el Modelo Vista Controlador como patrón de arquitectura web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Automatiza las tareas más comunes como generar controladores, vistas, entidades, conexiones con la base de datos entre otras permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Utiliza Doctrine como Mapeo Objeto-Relacional (ORM) para generar la capa de acceso a datos. Se puede ejecutar tanto en plataformas Unix (Linux, etc.) como en plataformas Windows.(Eguiluz 2014)

1.7. Herramientas para el desarrollo

1.7.1. IDE de Desarrollo. PHPStorm 8.0

PHPStorm 8 es un IDE de programación desarrollado por la empresa JetBrains, es uno de los entornos de programación más completos de la actualidad, permite editar código no sólo del lenguaje de programación PHP sino que posee soporte para múltiples lenguajes. Algunas de las características que posee son:

Editor de código PHP inteligente: el editor toma el código y entiende su estructura con bastante profundidad y soporta PHP desde la versión 5.3 hasta la 5.6 para proyectos más actualizados.

Análisis de la Calidad del Código: cientos de inspecciones se encargan de verificar el código mientras se escribe, analizando el proyecto completo. PHPDoc ayuda a arreglar y dar formato al código PHP*, Code Sniffer y Mess Detector* permiten la corrección rápida y ayudar a escribir un código limpio.

Variables de entorno: ejecutan varias tareas desde el mismo IDE, apoyo para el despliegue remoto, intérpretes de PHP remotos, bases de datos del lenguaje SQL, herramientas de líneas de código.

Depuración y prueba: la configuración de la depuración Zero hace muy fácil la depuración de tu aplicación, especialmente con la validación de configuración del depurador. El soporte PHPUnit permite desarrollar y ejecutar las pruebas unitarias desde el mismo IDE.

Editor HTML/CSS/Javascript: se admiten tecnologías avanzadas incluyendo HTML5, CSS, Sass, SCSS, *Less, Stylus, Compass, CoffeeScript, TypeScript, ECMAScript Harmony, Jade, Emmet,* y por supuesto *JavaScript* con refactorizaciones, depuración y pruebas unitarias.

Experiencia Multiplataforma: usa las mismas variables de entorno sobre *Windows, MacOS X o Linux* con la misma llave licencia única. (JetBrains 2015)

1.8. Servidor Apache HTTP Server 2.2

Un servidor web es un programa que implementa el protocolo HTTP (*hypertext transfer protocol*). Este protocolo está diseñado para transferir lo que llamamos hipertextos, páginas web o páginas HTML (*hypertext markup language*): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de música.(Dientuki 2008)

Apache es un servidor de código abierto para plataformas *Unix, Windows, Macintosh* y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPD 1.3, pero más tarde fue reescrito por completo.
Ventajas de Apache 2.2:

- Modular
- Código abierto
- Multi-plataforma
- Extensible

1.9. Conclusiones del capítulo.

Para llevar a cabo el desarrollo el bundle que integra los frameworks Qooxdoo y Symfony 2 se realizó un estudio de los conceptos principales asociados a ellos, se estudió algunas soluciones de *bundles* que integran otros *frameworks* como *ExtJS* y *jQuery* a *Symfony 2*. Esto conformó el punto de partida para lograr una solución a la problemática. Para guiar el proceso de desarrollo del *bundle* se seleccionó como

metodología de desarrollo de *software OpenUp* debido a que es una metodología ágil, centrada en equipos pequeños de trabajo, enfocada a un período corto de tiempo y genera una cantidad mínima de artefactos. El lenguaje de modelado previsto es UML 2.0 y como herramienta CASE Visual Paradigm en su versión 8.0. Por otra parte *JavaScript* y PHP son los lenguajes de programación ya que los *frameworks Qooxdoo* 4.1 y *Symfony* 2.3.7 utilizan estos lenguajes respectivamente. Como IDE de desarrollo fue seleccionado PHPStorm 8.0.

CAPÍTULO II. ANÁLISIS Y DISEÑO DEL SISTEMA

En este capítulo se representan todas las características que poseerá el futuro sistema. Se realizará el levantamiento de los requisitos funcionales y no funcionales, y a partir de ellos los casos de uso del sistema con sus respectivas descripciones textuales.

2.1. Modelo de dominio del sistema

El modelo de dominio también conocido como Modelo Conceptual es una representación visual de los conceptos u objetos del mundo real, significativos para un problema o área de interés. Este representa clases conceptuales del dominio del problema, conceptos del mundo real, no de los componentes de software. Una clase conceptual también conocida como entidad puede ser una idea o un objeto físico (símbolo, definición y extensión). (Larman 2004)

Teniendo en cuenta que los procesos no están bien definidos, es necesario describir el funcionamiento de la aplicación mediante una serie de conceptos, entidades y sus relaciones, agrupándose en un modelo de dominio con el fin de contribuir a la comprensión del contexto del sistema.

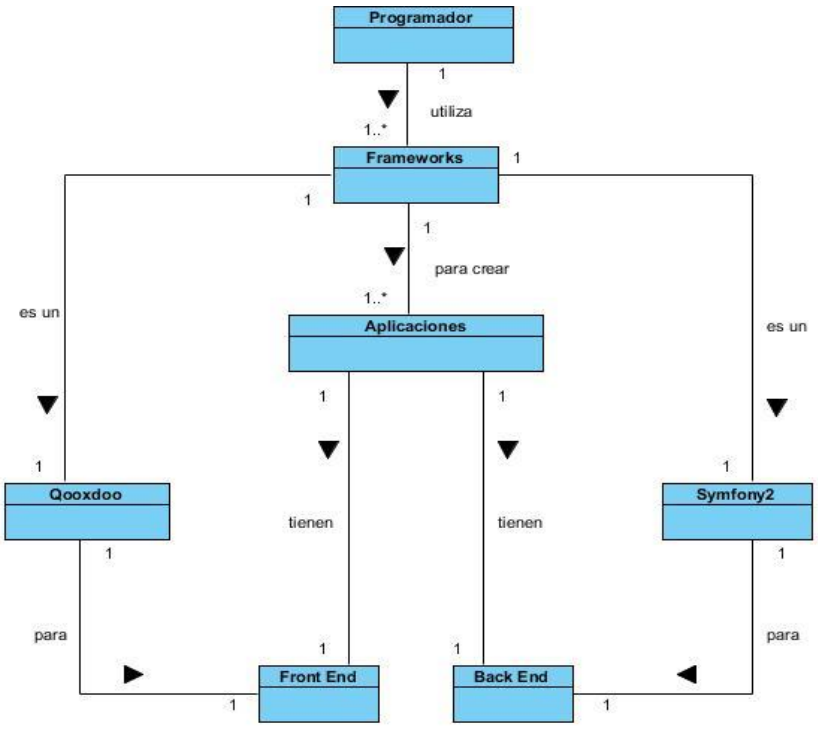

Fig. 5- Modelo de dominio del sistema

Clases Conceptuales del Dominio

Programador

Objeto que representa a la persona que utiliza los *frameworks Qooxdoo y Symfony 2* para la creación de aplicaciones web pero sin un mecanismo que le facilite la integración de los mismos.

Qooxdoo

Objeto que representa al *framework JavaScript* de desarrollo web *Qooxdoo.*

***Symfony* 2**

Objeto que representa al *framework* de desarrollo web *Symfony* 2.

Frameworks

Objeto que representa la herramienta que utiliza el programador para crear aplicaciones.

Aplicaciones

Objeto que representa a las acciones que se puede realizar con los frameworks.

Front End

Objeto que representa la parte de las vistas en una aplicación es decir la interfaces visuales.

Back End

Objeto que representa la parte que administra el sistema.

2.2. Requisitos

2.2.1. Requisitos funcionales

Los requisitos funcionales definen el comportamiento interno de un *software*, son condiciones que el sistema ha de cumplir. Estos muestran las funcionalidades que deben satisfacerse para cumplir con las especificaciones de *software*. (Sommerville 2007)

RF1- Generar bundle de *Qooxdoo*.

Entrada: namespace, nombre del bundle, directorio de salida del bundle, formato de configuración, generación de la estructura de directorios, confirmación de la generación, confirmación de la actualización del kernel, confirmación de la actualización de las rutas.

Descripción: Permite crear un bundle de Symfony 2 con un proyecto de Qooxdoo contenido.

Salida: Bundle de *Qooxdoo* generado.

RF2- Compilar bundle de *Qooxdoo*.

Entrada: namespace, opciones de compilación.

Descripción: Permite compilar un proyecto de *Qooxdoo* que este contenido en el *bundle* de *Symfony 2* donde se creó.
Salida: Proyecto de *Qooxdoo* compilado contenido en el *bundle* de *Symfony 2*.
RF3- Generar el CRUD para el bundle de Qooxdoo.
Entrada: nombre de acceso a la entidad, formato de configuración, prefijo, confirmación de escritura de los archivos base, confirmación de la generación del CRUD, confirmación de la actualización de las rutas.
Descripción: Permite generar el CRUD a una entidad previamente creada en un *bundle* de *Symfony 2*.
Salida: CRUD generado para la entidad escogida.

2.2.2. *Requisitos no funcionales*

Los requisitos no funcionales son propiedades que hacen al producto atractivo, usable, rápido o confiable. Se refieren a todos los requisitos que ni describen información a guardar, ni funciones a realizar. Además se conocen como un conjunto de características de calidad, que es necesario tener en cuenta al diseñar e implementar el software. (Sommerville 2007)

RnF1: El sistema debe ser utilizado por personas con conocimientos de los *frameworks Qooxdoo* y *Symfony 2* debido a que tendrá un alto grado de complejidad.
RnF2: El sistema será multiplataforma, se requiere tener instalado para su correcto funcionamiento PHP en su versión 5.3 o superior y el Servidor Apache 2.2.
RnF3: El sistema será implementado utilizando los framework de desarrollo Qooxdoo 4.1 y Symfony2
RnF4: Las características mínimas recomendables del ordenador para que el sistema pueda funcionar correctamente son:

- Capacidad libre en disco duro de 300 MB
- Memoria RAM ≥ 1 GB
- Microprocesador Celeron con 2.0 GHz de velocidad.

RnF5: La aplicación contará antes de su puesta en marcha con un período de pruebas y contará con un manual de usuario.

2.3. Modelo de Casos de Uso del Sistema (CUS)

El diagrama de casos de uso representa la forma en cómo un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Tabla 4- Descripción Actores del Sistema

Actor	Descripción
Programador	Persona que trabaja con el *bundle Qooxdoo-Bundle* para el desarrollo de aplicaciones web.

2.3.1. *Diagrama de casos de uso del sistema*

El diagrama de casos de uso representa la interacción entre el cliente (Actor) y el sistema que se encuentra en desarrollo, además del orden en que los elementos (casos de uso) interactúan entre sí.

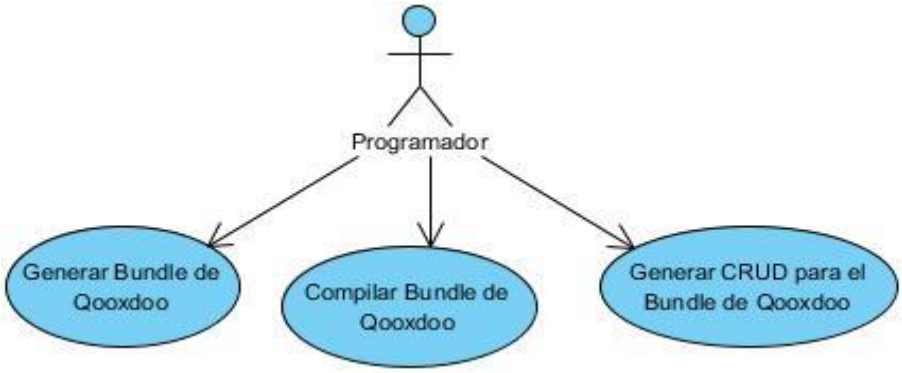

Fig. 6- Diagrama de Casos de Uso del Sistema

CUS 1: Generar *bundle* de *Qooxdoo*: este Caso de Uso tiene la finalidad de generar un *bundle* que contenga lo estructura de *Qooxdoo*.

CUS 2: Compilar *bundle* de *Qooxdoo*: este Caso de Uso tiene la final compilar un *bundle* de *Qooxdoo*.

CUS 3: Generar CRUD para un *bundle*: este Caso de Uso tiene la final generar un CRUD de una entidad de un *bundle* de *Qooxdoo*.

2.3.2. *Especificación de casos de uso*

CU1. Crear *bundle* de *Qooxdoo*

Tabla 5-- Descripción del CU Generar bundle de *Qooxdoo*

Objetivo	Crear un comando que permita generar un proyecto de *Qooxdoo* en *Symfony* 2

Actores	Programador(Inicia)
Resumen	Comando que permite la creación de *bundle*s que contengan la estructura de *Qooxdoo*.
Complejidad	Alta
Prioridad	Crítico
Precondiciones	El usuario debe haber declarado la variable del sistema *QOOXDOO*_PATH donde la misma debe contener la dirección donde se encuentra el SDK de *Qooxdoo*. El usuario debe tener conocimiento acerca de los *framework Qooxdoo y Symfony 2.* El usuario debe tener abierta la consola de *Symfony* del proyecto donde se va crear el *bundle* de *Qooxdoo*.
Postcondiciones	Se crea un *bundle* que contiene la estructura de *Qooxdoo*.
Flujo de eventos	
Flujo básico Crear *bundle* de *Qooxdoo*	

	Actor	**Sistema**
1.	Escribe en consola el comando *Qooxdoo*:generate-*bundle*	Muestra la interfaz de creación de un *bundle* la cual te pide el namespace del *bundle* a crear(Este namespace debe estar escrito de la siguiente forma Ej. Acme/Demo*Bundle*).
2.	Escribe el namespace del *bundle* a crear	Valida el namespace entrado por el usuario y a continuación pide el nombre del *bundle* (Ej. Demo*Bundle*).

3.	Escribe el nombre del *bundle* a crear	Valida el nombre entrado por el usuario y a continuación pide la dirección donde guardar el *bundle* (por defecto lo guarda en /src).
4.	Escribe la dirección donde guardar el *bundle*	Valida la dirección donde se guardara el *bundle* y pide el formato de configuración este puede ser: yml, xml, php o annotation. Por defecto viene la acción annotation.
5.	Escribe el formato de configuración.	Valida el formato de configuración y pide al usuario si quiere generar toda la estructura de directorios (por defecto está la configuración no).
6.	Escribe la opción deseada.	Valida la opción y muestra la opción de actualizar automáticamente la ruta del *bundle* en el kernel de *Symfony* (por defecto la opción está en yes).
7.	Escribe la opción deseada.	Valida la opción y muestra la opción de actualizar automáticamente las rutas del sistema (por defecto está en yes).
8.	Escribe la opción deseada.	Validad la opción y el genera el *bundle* con las opciones pasadas por el usuario, mostrando un mensaje de que el *bundle* ha sido generado satisfactoriamente y terminando el caso de uso.
Flujos alternos		
2a. Namespace del *bundle* incorrecto		
	Actor	**Sistema**
1.		Envía un mensaje de error alertando al usuario de que el namespace del *bundle* escrito no es correcto o no existe.

3a. Nombre del *bundle* incorrecto		
	Actor	**Sistema**
1.		Envía un mensaje de error alertando al usuario de que el nombre del *bundle* no es correcto.
4a. Dirección donde guardar el *bundle* incorrecta		
	Actor	**Sistema**
1.		Envía un mensaje de error alertando al usuario de que la dirección donde guardar el *bundle* es incorrecta.
5a. El formato de configuración es incorrecto		
	Actor	**Sistema**
1.		Envía un mensaje de error alertando al usuario de que el formato de configuración es incorrecto
6a, 7a, 8a. La opción es incorrecta		
	Actor	**Sistema**
1.		Envía un mensaje de error alertando al usuario de que la opción seleccionada es incorrecta

Relaciones	**CU Incluidos**	-
	CU Extendidos	-

Requisitos no funcionales	-
Asuntos pendientes	-

2.4. Patrones de Arquitectura

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de *software*. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Los patrones de arquitectura representan el nivel más alto en el sistema.

Patrón n capas

El estilo arquitectural en capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan. (Macias 2013)

Características

- Descomposición de los servicios de forma que la mayoría de interacciones ocurre solo entre capas vecinas.
- Las capas de una aplicación pueden residir en la misma máquina o pueden estar distribuidos entre varios equipos.
- Los componentes de cada capa se comunican con los componentes de otras capas a través de interfaces bien conocidos.
- Cada nivel agrega las responsabilidades y abstracciones del nivel inferior. (Olea 2014)

Para el diseño de este *bundle* se escoge el patrón n-capas dividiendo el mismo en 3 capas, las cuales se enuncian a continuación:

Capa Command: es la capa de más alto nivel la cuál es la encargada de interactuar con el usuario y enviarle las peticiones realizadas por el mismo a la Capa *Generator.*

Capa Generator: es la encargada de generar los archivos a partir de las peticiones realizadas por el usuario, para esto le envía a las plantillas de la Capa *Resource* los datos entrados por el mismo. En esta también se encuentra incluido el paquete *Manipulator* que se encarga de gestionar las rutas de los bundles y la actualización del kernel.

Capa Resource: tiene todas las plantillas las cuales van a contener la estructura necesaria para conformar los *bundle*s con estructuras de *Qooxdoo*.

2.5. Modelo de diseño

Expande y detalla el modelo de análisis tomando en cuenta todas las implicaciones y restricciones técnicas, donde se distinguen tres capas que son el diseño del sistema, el diseño de objetos y el diseño de la persistencia. Permite experimentar y visualizar el sistema que se construirá. Donde se puede razonar acerca de los requerimientos del sistema con un cliente o usuario final, también a los detalles y la vista que presenta un modelo al usuario.

2.5.1. Diagramas de Clases del Diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases del software y de las interfaces de una aplicación. Normalmente contienen información acerca de las clases, asociaciones, atributos, métodos y dependencias.(Larman 2004)

La figura 7 representa la estructura de paquetes del bundle QooxdooSymfony2, este diagrama está estructurado utilizando el patrón arquitectónico n capas, específicamente en 3 capas las cuales son: capa command, capa generator y la capa resource.

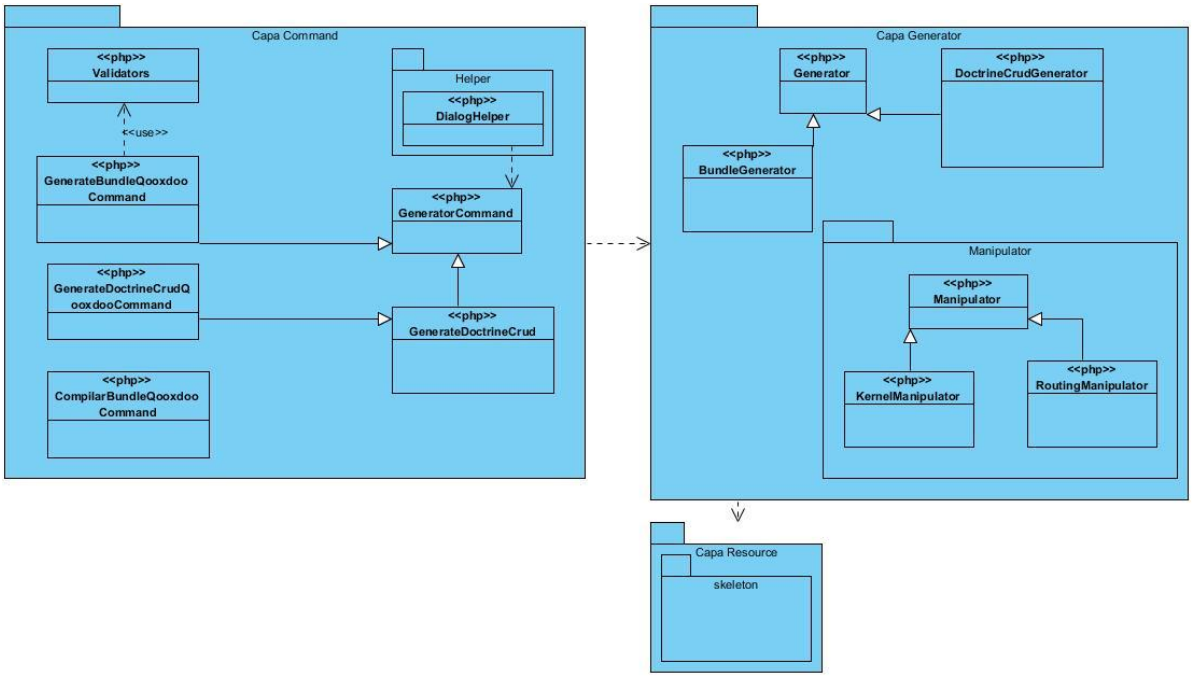

Fig. 7- Diagrama de paquete del bundle QooxdooSymfony2

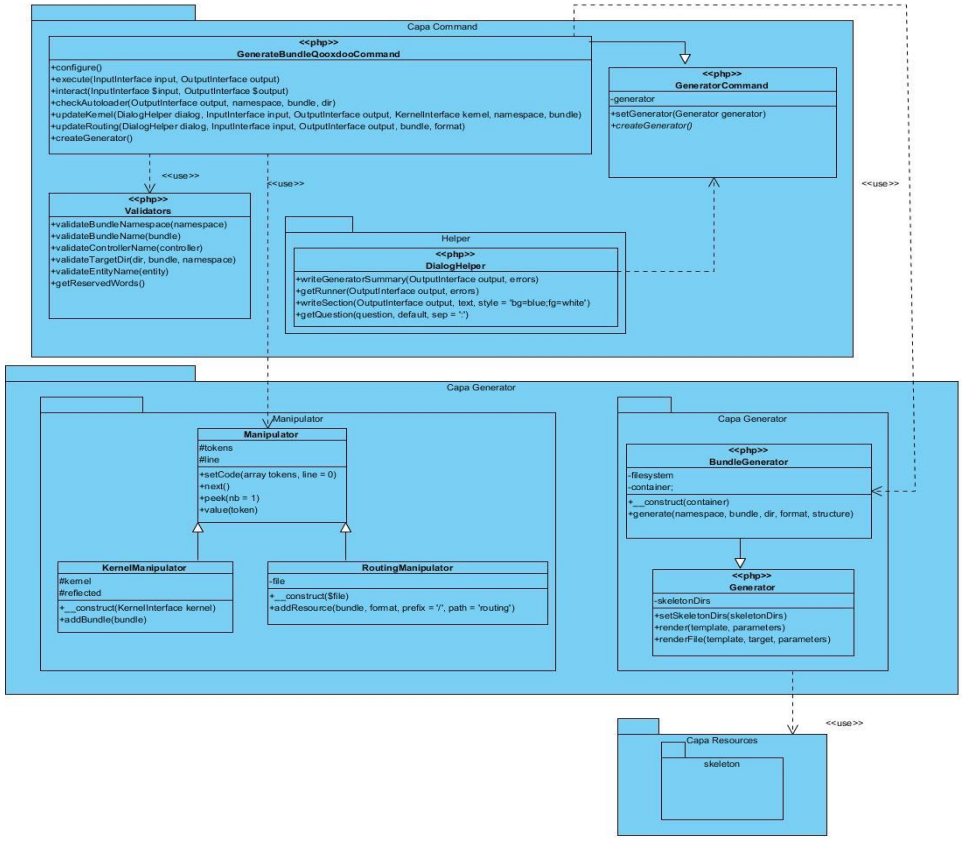

Fig. 8- Diagrama de clase del diseño del CU Generar bundle de Qooxdoo

Principales clases

- **Generate*BundleQooxdoo*Command:** clase que extiende del **GeneratorCommand** la cual es la responsable de crear el comando para generar un *bundle*, en esta se definen los parámetros del comando, la misma interactúa con los usuarios mediante mensajes en la consola para obtener los parámetros de entrada y validarlos utilizando la clase **Validators**.
- **GeneratorCommand:** clase padre que extiende de la clase de *Symfony* 2 **ContainerAwareCommand**, la misma define la ruta de los directorios que se van a copiar al *bundle* generado y se los pasa a la clase **Generator**.
- **DialogHelper:** clase encargada de mostrar los mensajes en pantalla.
- **Validators:** clase encargada de hacer las validaciones de los valores entrados por parámetro.
- **Generator:** clase que modifica los archivos que se van a copiar en el *bundle* con los valores pasados por parámetros por el usuario.
- ***Bundle*Generator:** clase que extiende del **Generator** y contiene todas las direcciones de los

archivos que se van a generar.

- **Manipulator:** clase padre encargada de manipular los archivos de configuración de *Symfony* 2.
- **RoutingManipulator:** clase que hereda de **Manipulator** encargada de manipular las rutas de *Symfony* 2.
- **KernelManipulator:** clase que hereda de **Manipulator** encargada de manipular el AppKernel de *Symfony* 2.
- **Paquete Skeleton:** en él se encuentran todas las plantillas de los archivos a copiar en el *bundle* que se va a crear.

2.5.2. *Patrones de diseño*

GRASP (*General Responsibility Assignment Software Patterns*)

- **Patrón Experto:** el patrón experto describe a una clase que cuenta con toda la información necesaria para asignar una responsabilidad. Este se evidencia en la clase **Generate*BundleQooxdoo*Command**, esta contiene la información necesaria de un bundle de Qooxdo en Symfony 2.

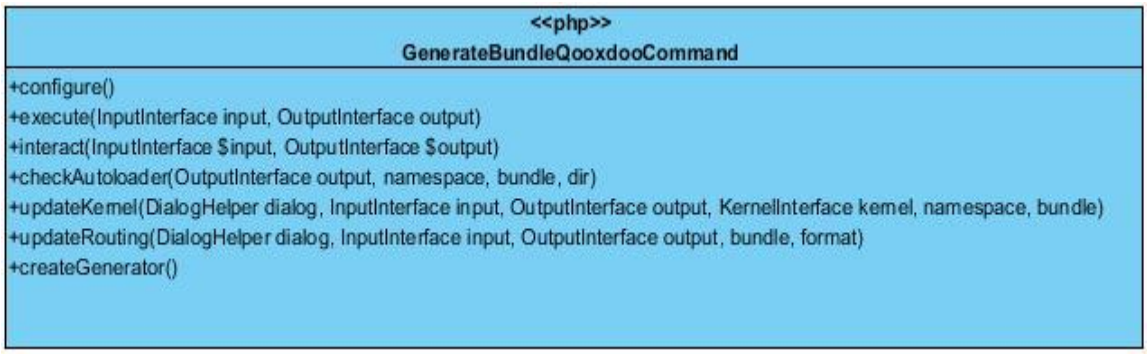

Fig. 9- Patrón Experto

- **Patrón Creador:** el patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos.(Larman 2004)

- Este patrón se ve reflejado en la clase **Generate*BundleQooxdoo*Command** puesto que esta es la encargada de crear la instancia de la clase ***Bundle*Generator.**

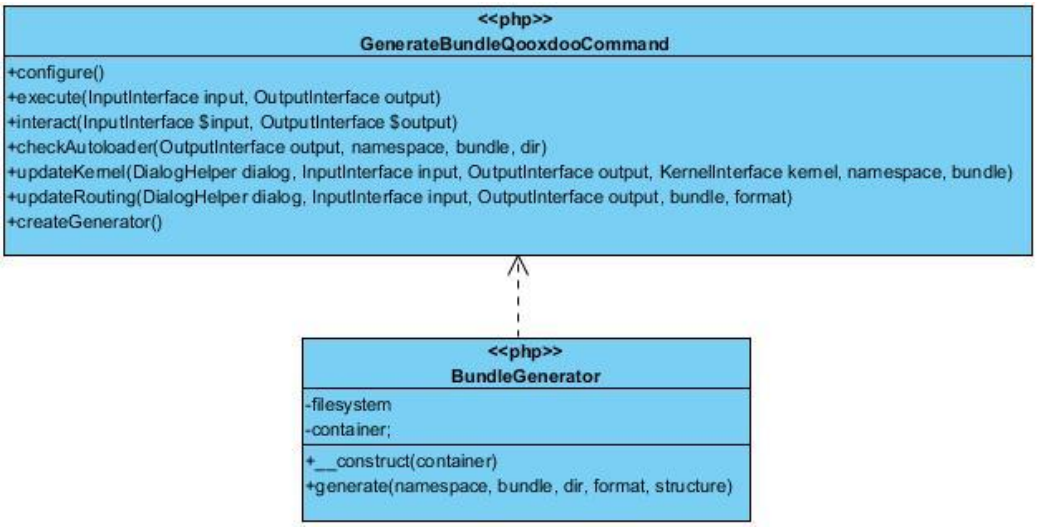

Fig. 10-Patrón Creador

- **Patrón Polimorfismo:** cuando por el tipo varían las alternativas o comportamientos afines, las responsabilidades del comportamiento se asignaran mediante operaciones polimórficas a los tipos en que el comportamiento presenta variantes.(Larman 2004) Las clases ***Bundle*Generator y DoctrineCrudGenerator** son hijas de la clase **Generator,** las mismas reimplementan sus métodos e implementan nuevos.

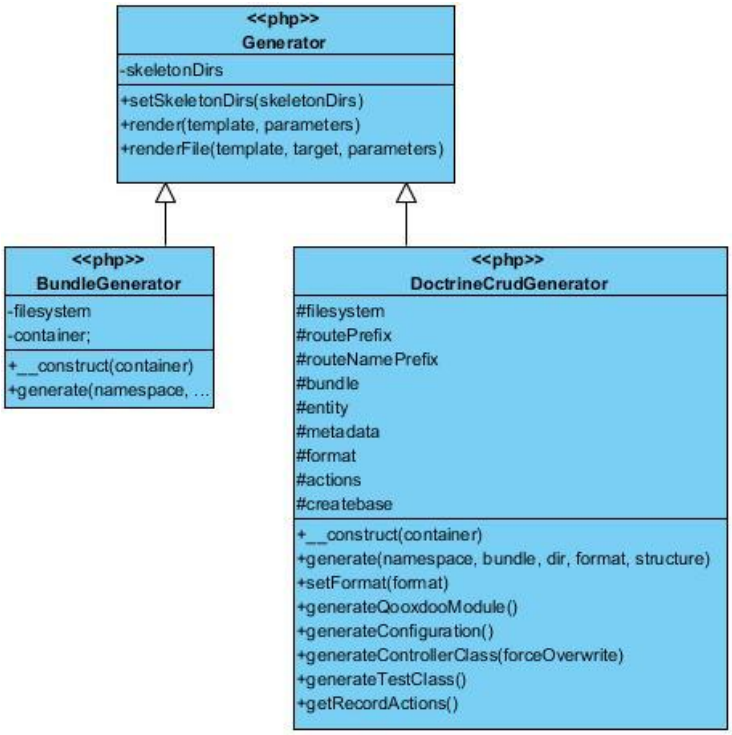

Fig. 11- Patrón Polimorfismo

GOF (Gang of Four)

- **Patrón *Builder*:** es usado para permitir la creación de una variedad de objetos complejos desde un objeto fuente, el cual se compone de una variedad de partes que contribuyen individualmente a la creación de cada objeto complejo a través de un conjunto de llamadas a interfaces comunes de la clase Abstract Builder. Con la clase GeneratorCommand se crean todas las clases que definen algún comando en el bundle.

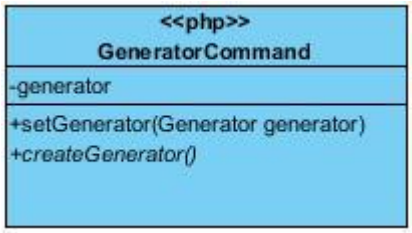

Fig. 12- Patrón Builder

- **Patrón *Command*:** encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer

la operaciones. Este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además se facilita la parametrización de los métodos.(Thompson 2009)

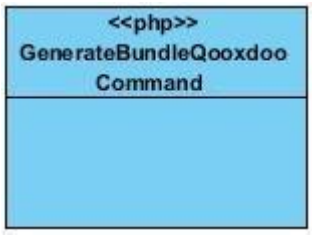

Fig. 13- Patrón Command

Esta clase representa al comando de generar *bundle*s, es la encargada de la interacción con el usuario para crear los objetos necesarios a partir de los datos entrados por los usuarios para generar los bundles con estructura de *Qooxdoo*.

2.6. Conclusiones del capítulo

En el presente capítulo se realizó una representación visual de clases conceptuales del entorno real de los objetos del proyecto a través del modelo de dominio. Se identificaron los 3 requisitos funcionales agrupados en 3 casos de usos y 5 requisitos no funcionales que se deben cumplir para el correcto funcionamiento del *bundle*. Se seleccionó el patrón de diseño n-capas y la representación del mismo. Se identificaron los patrones de diseño GRASP y GOF para un mejor diseño del *bundle*. Se realizó el diagrama de clases del diseño para la representación de las clases y las relaciones entre ellas.

CAPÍTULO III: IMPLEMENTACIÓN Y PRUEBA DEL *BUNDLE QOOXDOO-SYMFONY 2*

En este capítulo se muestra el modelo de implementación como resultado del diseño anteriormente desarrollado. Como parte del mismo se presenta el diagrama de componentes del CU Generar *bundle* de *Qooxdoo* y los estándares de codificación utilizados para el desarrollo de la aplicación. Además se describen las pruebas a realizar, con el objetivo de comprobar el correcto funcionamiento del *bundle Qooxdoo-Symfony* 2 en distintos momentos del ciclo de vida del software.

3.1. Modelo de implementación

3.1.1 *Diagrama de componentes*

Un diagrama de componente es utilizado para representar la separación de un sistema de *software* en componentes físicos y mostrar las dependencias entre estos. Estos componentes incluyen: archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables y paquetes. (Jacobson 2010)

En la Fig. 14 que se muestra a continuación se presenta el diagrama de componente del CU Generar *bundle* de *Qooxdoo*. El mismo está dividido en 4 paquetes:

- **Command:** agrupa las clases que se utilizan para crear los comandos.
- **Generator**: agrupa las clases que se utilizan para generar los archivos del *bundle* a crear.
- **Manipulator**: contiene las clases que manipulan las rutas y el kernel de *Symfony* 2.
- **Resource**: contiene las plantillas que se van a generar a partir de las opciones entradas por los usuarios.

Además muestra el subsistema *Symfony* 2 con el cual se relacionan todos los paquetes de la aplicación.

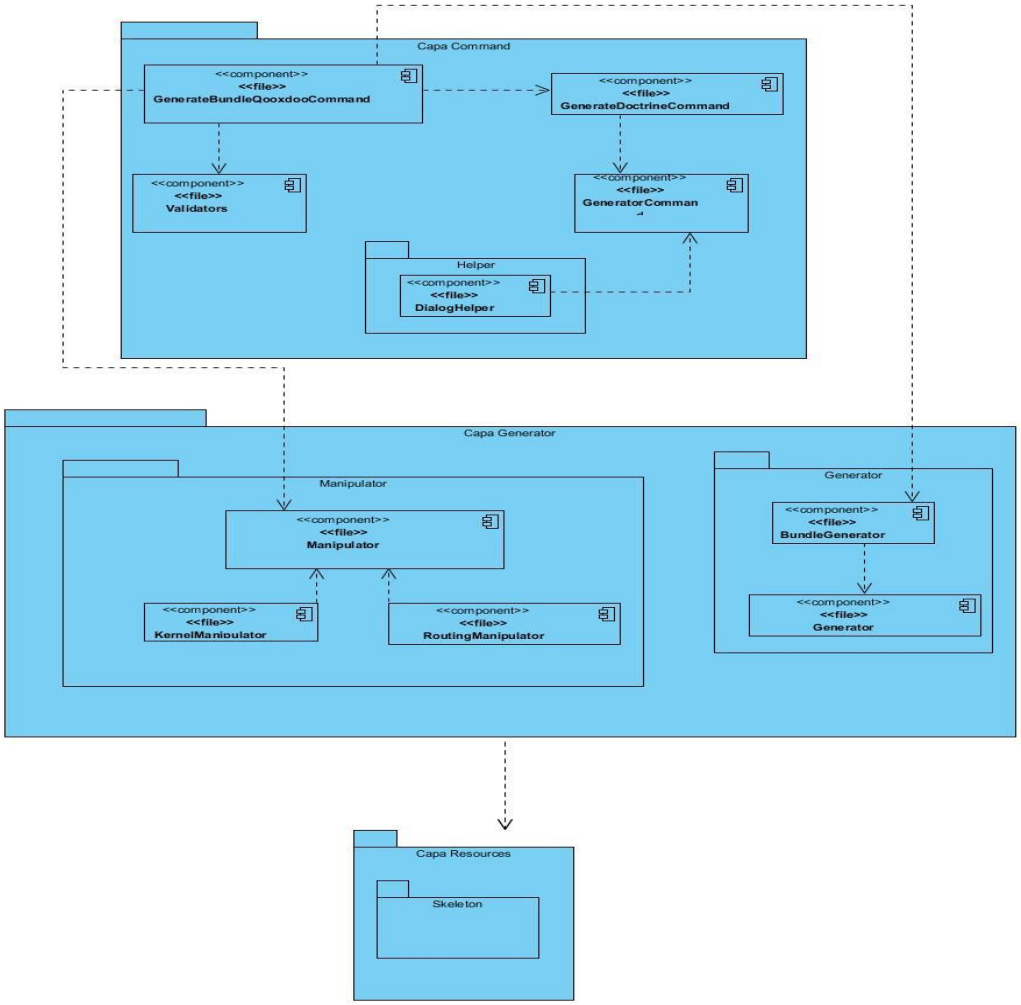

Fig. 15- Diagrama de Componentes del CU Crear *bundle* de *Qooxdoo*

3.3. Código Fuente.

3.3.1 Estándares de codificación

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Un estándar de codificación completo comprende todos los aspectos de la generación de código. Un código fuente completo debe reflejar un estilo armónico, como si un único programador hubiera escrito todo el código de una sola vez. Posibilita que un equipo de programadores mantenga un código de calidad sobre el que se efectuarán luego revisiones.

El estándar de codificación seleccionado para el desarrollo de la aplicación es el *CamelCase.*

CamelCase: es un estilo de escritura que se aplica a frases o palabras compuestas, dentro de este estilo existen dos tipos:

- **UpperCamelCase:** las palabras que forman el nombre se escriben juntas y la primera letra de cada una de ellas en mayúscula.

```php
namespace UCI\QooxdooBundle\Generator;

use Symfony\Component\Filesystem\Filesystem;
use Symfony\Component\HttpKernel\Bundle\BundleInterface;
use Doctrine\ORM\Mapping\ClassMetadataInfo;

/**
 * Generates a CRUD controller.
 *
 * @author Fabien Potencier <fabien@symfony.com>
 */
class DoctrineCrudGenerator extends Generator
{
```

Fig. 16- Estilo de código UpperCamelCase

- **lowerCamelCase:** las palabras que forman el nombre se escriben juntas, la primera letra de la primera palabra en minúscula y del resto de las palabras, la primera letra en mayúscula.

```php
class DoctrineCrudGenerator extends Generator
{
    protected $filesystem;
    protected $routePrefix;
    protected $routeNamePrefix;
    protected $bundle;
    protected $entity;
    protected $metadata;
    protected $format;
    protected $actions;
    protected $createbase;

    /**
     * Constructor.
     *
     * @param Filesystem $filesystem A Filesystem instance
     */
    public function __construct(Filesystem $filesystem)
    {
        $this->filesystem  = $filesystem;
    }
```

Fig. 17- Estilo de código lowerCamelCase

Estilo de codificación utilizado

- El código PHP debe estar delimitado siempre por la forma completa de las etiquetas PHP estándar: <?php?>.

- Los nombres de las clases y los ficheros deben ser escritos utilizando el estándar de codificación *UpperCamelCase*.
- Los nombres de las funciones deben ser lo suficientemente elocuentes como para describir su propósito y comportamiento. Los nombres de las variables deben ser siempre claros, evidentes y prácticos para describir los datos que el desarrollador pretende almacenar en ellas. Sus nombres deberán estar escritos utilizando el estándar de codificación *lowerCamelCase.*
- Los comentarios de implementación están delimitados por /*...*/ o //.
- Todas las funciones y clases estarán comentadas para explicar el propósito de las mismas.

3.4. Pruebas de Software.

Las Pruebas de *software* pueden definirse como “el proceso de evaluación de un producto desde un punto de vista crítico, donde el "probador" (persona que realiza las pruebas) somete al producto a una serie de acciones indagadoras, y el producto responde con su comportamiento como reacción”. (Preesman 2011)

3.4.1. Niveles de Prueba

A la hora de evaluar dinámicamente un sistema de *software* se debe comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el *software* en su conjunto. Las pruebas se aplican durante todo el ciclo de desarrollo del *software* para diferentes objetivos y en distintos niveles de trabajo a continuación se mencionarán los niveles que se aplicarán:

- Pruebas de Desarrollador
- Pruebas de Integración
- Pruebas de Aceptación

Los niveles que se han seleccionado para llevar a cabo en la aplicación son:

- **Pruebas de Desarrollador**

  Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas pruebas han sido consideradas solo para la prueba de unidad, aunque en la actualidad en algunos casos pueden ejecutar pruebas de integración. Se recomienda que estas pruebas cubran más que las pruebas de unidad. (Preesman 2011)

- **Pruebas 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. Es el proceso de combinar y probar múltiples componentes juntos. El objetivo es tomar los componentes probados en unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Se llama integración incremental cuando el programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y corregir. (Preesman 2011)

- **Pruebas de Aceptación**

Las pruebas de aceptación se realizar para permitir que el cliente valide todos los requisitos. Las realiza el usuario final en lugar del responsable del desarrollo del sistema, una prueba de aceptación puede ir desde un informal paso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. De hecho, la prueba de aceptación puede tener lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema. (Preesman 2011)

3.4.2. Métodos de Prueba

- **Método de pruebas de caja blanca**

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba.

La pruebas de caja blanca intentan garantizar que: se ejercite por lo menos una vez todos los caminos independientes de cada módulo; ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas; ejecuten todos los bucles en sus límites; y ejerciten las estructuras internas de datos para asegurar su validez. (Preesman 2011)

Técnica camino básico

Una de las técnicas de prueba de caja blanca más usada es el camino básico, la cual determina la complejidad ciclomática de una porción de código. La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa el camino básico, 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. (Preesman 2011)

- **Método de pruebas de caja negra**

La prueba de la caja negra es un método de diseño de casos de prueba que se centra en los requisitos funcionales del software, por lo que se llevan a cabo sobre la interfaz del software. El objetivo es demostrar que las funciones del software son operativas, que las entradas se acepten de forma adecuada y que la integridad de la información externa se mantenga. La prueba de caja negra intenta identificar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a bases de datos externas, errores de rendimiento y errores de inicialización y de terminación. (Preesman 2011)

Técnica particiones equivalentes

Una de las técnicas de prueba de caja negra más usada es la partición equivalente, la cual divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.(Preesman 2011)

3.4.3. Diseño de casos de prueba

Se trata de diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y de tiempo. En la prueba de caja blanca se realiza un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en varios puntos.

En la prueba de la caja negra, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

3.4.4 Proceso de prueba de la herramienta

Con la intención de poder descubrir y corregir el máximo de errores posibles en la herramienta desarrollada, se decidieron realizar las pruebas siguientes. A nivel de desarrollador se aplicó el método de prueba de caja blanca utilizando la técnica de camino básico y pruebas unitarias utilizando las funcionalidades que ofrece el *framework Symfony* 2 para el desarrollo de pruebas de este tipo. A nivel de integración se aplicó el método de caja negra y la técnica de particiones equivalentes para verificar el correcto funcionamiento de la aplicación. A nivel de aceptación se utilizó las pruebas de aceptación de tipo Alfa.

3.4.5. Aplicación de las pruebas de caja blanca.

Técnica de camino básico.

CAPÍTULO III: IMPLEMENTACIÓN Y PRUEBA DEL *BUNDLE QOOXDOO*-SYMFONY2

Como se mencionó en acápites anteriores, una de las técnicas de prueba de caja blanca es la de camino básico, que se aplica a fragmentos de código. En el proceso de prueba para validar la herramienta desarrollada se decidió aplicar esta técnica a los métodos de la clase del CU Generar *bundle* de *Qooxdoo*. En el cual se identificaron 17 bloques de ejecución, los cuales fueron enumerados para ser reconocidos. Además se determinó el número de caminos independientes máximo para asegurar que se ejecuta cada sentencia al menos una vez mediante el cálculo por las diferentes vías de la complejidad ciclomática. A continuación en las Figuras 15 y 16 se ejemplifica algunos de los pasos realizados para calcular la complejidad ciclomática.

- El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
- La complejidad ciclomática, V(G), de un grafo de flujo G se define como.

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

- La complejidad ciclomática, V(G), de un grafo de flujo G también se define como V(G) = P + 1 donde P es el número de nodos predicado contenidos en el grafo de flujo G.

```
protected function execute(InputInterface $input, OutputInterface $output)
{
    $dialog = $this->getDialogHelper();
    if ($input->isInteractive()) {
        if (!$dialog->askConfirmation($output, $dialog->getQuestion('¿Quieres confirmar la generación?', 'yes',''), tr
            $output->writeln('<error>Command aborted</error>');
            return 1;
        }
    }
    foreach (array('namespace', 'dir') as $option) {
        if (null === $input->getOption($option)) {
            throw new \RuntimeException(sprintf('La "%s" opcion debe ser proporcionada.', $option));
        }
    }
    $namespace = Validators::validateBundleNamespace($input->getOption('namespace'));
    if (!$bundle = $input->getOption('bundle-name')) {
        $bundle = strtr($namespace, array('\\' => ''));
    }
    $bundle = Validators::validateBundleName($bundle);
    $dir = Validators::validateTargetDir($input->getOption('dir'), $bundle, $namespace);
    if (null === $input->getOption('format')) {
        $input->setOption('format', 'annotation');
    }
    $format = Validators::validateFormat($input->getOption('format'));
    $structure = $input->getOption('structure');
    $dialog->writeSection($output, 'Bundle de Qooxdoo Generado');

    if (!$this->getContainer()->get('filesystem')->isAbsolutePath($dir)) {
        $dir = getcwd().'/'.$dir;
    }
    $namebundle=$input->getOption('bundle-name');
    $name=explode("Bundle",$namebundle);
    exec('python $QOOXDOO_PATH/create-application.py -n '.$name[0]);
    $generator = $this->getGenerator();
    $generator->generate($namespace, $bundle, $dir, $format, $structure);
    $output->writeln('El codigo del bundle se ha generado: <info>OK</info>');
    $errors = array();
    $runner = $dialog->getRunner($output, $errors);
    $this->getContainer()->get('filesystem')->remove('./'.$name[0]);
    $runner($this->checkAutoloader($output, $namespace, $bundle, $dir));
    $runner($this->updateKernel($dialog, $input, $output, $this->getContainer()->get('kernel'), $namespace, $bundle));

    $runner($this->updateRouting($dialog, $input, $output, $bundle, $format));
    $dialog->writeGeneratorSummary($output, $errors);
}
```

Fig. 18- Código fuente del método execute

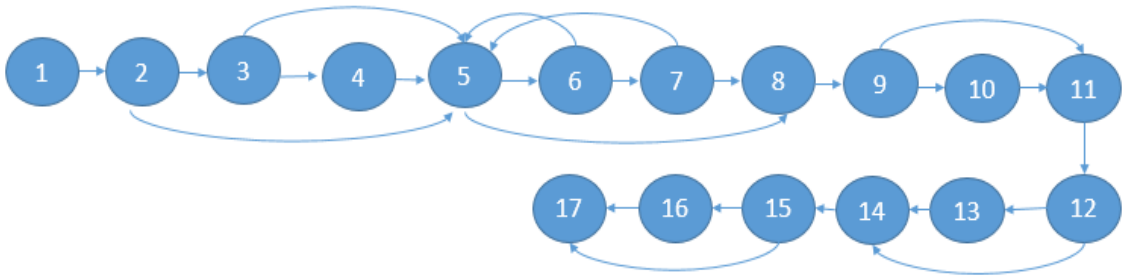

Fig. 19- Grafo de flujo del método execute

Después de representado el grafo de flujo se aplicaron las tres formas para calcular la complejidad ciclomática. Primeramente se identificaron 8 regiones en el grafo. Se utilizó la fórmula V (G) = A - N + 2, para la cual se encontraron 23 artistas y 17 nodos, por lo tanto: V (G) = 23 - 17 + 2, quedando V (G) = 8 y por último mediante la fórmula V (G) = P + 1, se detectaron 7 nodos predicados, resultando V (G) = 7 + 1, donde V(G) = 8. De esta forma se puede comprobar que las variantes para calcular la complejidad ciclomática arriban al mismo valor.

3.4.6. Pruebas unitarias o tests unitarios.

Los tests unitarios prueban pequeñas partes del código de la aplicación verificando que estas funcionan tal y como debería hacerlo. Idealmente, estos segmentos del código son la parte más pequeña posible que se pueda probar. Por convención, cada test unitario y funcional de *Symfony* 2 se define en una clase cuyo nombre acaba en Test y se encuentra dentro del directorio Tests/ del *bundle*. Además, se recomienda utilizar dentro de Tests/ la misma estructura de directorios del elemento que se quiere probar. Si se prueba por ejemplo el controlador por defecto del *bundle* Oferta, su test debería crearse en src/Cupon/Oferta*Bundle*/Tests/Controller/DefaultControllerTest.php.(Eguiluz 2014)

Para el desarrollo de los test unitarios en el *bundle QooxdooSymfony* 2 se aplicaron para su representación en este documento en los controladores que genera el CU Generar CRUD para *bundle* de *Qooxdoo* y a partir de esto se realizaron los métodos por de los cuales se basaran los test unitarios.

3.4.7. Casos de Prueba de caja negra.

Un caso de prueba se diseña según las funcionalidades descritas en los casos de uso. El propósito que se persigue con este artefacto es lograr una comprensión común de las condiciones específicas que la solución debe cumplir. Se parte de la descripción de los casos de uso del sistema, como apoyo para las revisiones. Cada planilla de caso de prueba recoge la especificación de un caso de uso, dividido en secciones y escenarios, detallando las funcionalidades descritas en él y describiendo cada variable que recoge el caso de uso en cuestión. Se efectuaron los casos de pruebas a los 3 casos de uso del sistema, plasmándose en la documentación del proyecto. (Ver planilla Diseño de Casos de Prueba)

Tabla 6- Variables de entradas del caso de prueba del caso de uso Generar bundle de *Qooxdoo.*

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	namespace	N/A	No.	Representa el namespace del *bundle* a crear
2	dir	N/A	No.	Directorio donde se creará el *bundle*
3	*bundle*-name	N/A	No.	Nombre que tendrá el *bundle* a crear
4	format	N/A	No.	Formato a utilizar en los archivos de configuración del *bundle*
5	structure	N/A	Si.	Identificador del usuario que creo la categoría.

Tabla 7- Caso de prueba del caso de uso Generar bundle de *Qooxdoo*

scenario	Descripción	namespace	dir	*bundle*-name	format	structure	Respuesta del sistema	Flujo central
EC 1.1 Escribir_campos_True	En este escenario se realiza la creación de un *bundle* con todos los campos pasados correctamente.	V UCI/Ejemplo*Bundle*	V /var/www/html/*QooxdooSymfony* 2/src	V Ejemplo *Bundle*	V yml	V yes	Se crea un *bundle* con los valores pasados por parámetros	A través de la consola de *Symfony* ejecutar el comando *Qooxdoo*:generate:*bundle*
EC 1.2 Campo_namesp	En este escenario se intenta crear un *bundle*	I	NA	NA	NA	NA	Muestra el mensaje the	A través de la consola de *Symfony* ejecutar el

ace_Blanco	con el namespace en blanco						namespace must end with *Bundle*	comando *Qooxdoo*:generate:*bundle*
EC 1.3 Campo_namespace_false	En este escenario se intenta crear un *bundle* con el namespace incorrecto	I	NA	NA	NA	NA	Muestra el mensaje the namespace must end with *Bundle*	A través de la consola de *Symfony* ejecutar el comando *Qooxdoo*:generate:*bundle*
		UCI						
EC 1.4 Campo_format_blanco	En este escenario se intenta crear un *bundle* con el formato en blanco	V	V	V	NA	NA	Muestra el mensaje format "" is not suported	A través de la consola de *Symfony* ejecutar el comando *Qooxdoo*:generate:*bundle*
		UCI/A*bundle*	/var/www/html/*QooxdooSymfony* 2/src	A*bundle*				
EC 1.5 Campo_format_incorrecto	En este escenario se intenta crear un *bundle* con el formato en incorrecto	V	V	V	I	NA	Muestra el mensaje format "mal" is not suported	A través de la consola de *Symfony* ejecutar el comando *Qooxdoo*:generate:*bundle*
		UCI/A*bundle*	/var/www/html/*QooxdooSymfony* 2/src	A*bundle*	mal			

3.4.8. Resultados de las Pruebas

Luego de aplicados cada uno de los niveles de prueba se realiza una evaluación de los resultados arrojados, donde se recogen de manera resumida las principales no conformidades encontradas.

Después de realizar las pruebas unitarias se obtuvo como resultado que las funcionalidades identificadas cumplían con la función para la cual fueron definidas; además de comprobarse su correcto funcionamiento, por lo que para dichas pruebas se encontraron 0 no conformidades.

Se realizaron las pruebas funcionales con el método de caja negra y la técnica de particiones equivalentes detectándose 13 no conformidades durante 3 iteraciones agrupándolas en 3 tipos Ortográficas, Funcionales y de Validación, las cuales fueron resueltas en la medida en que se detectaron. A continuación se muestra un gráfico que ilustra este resultado.

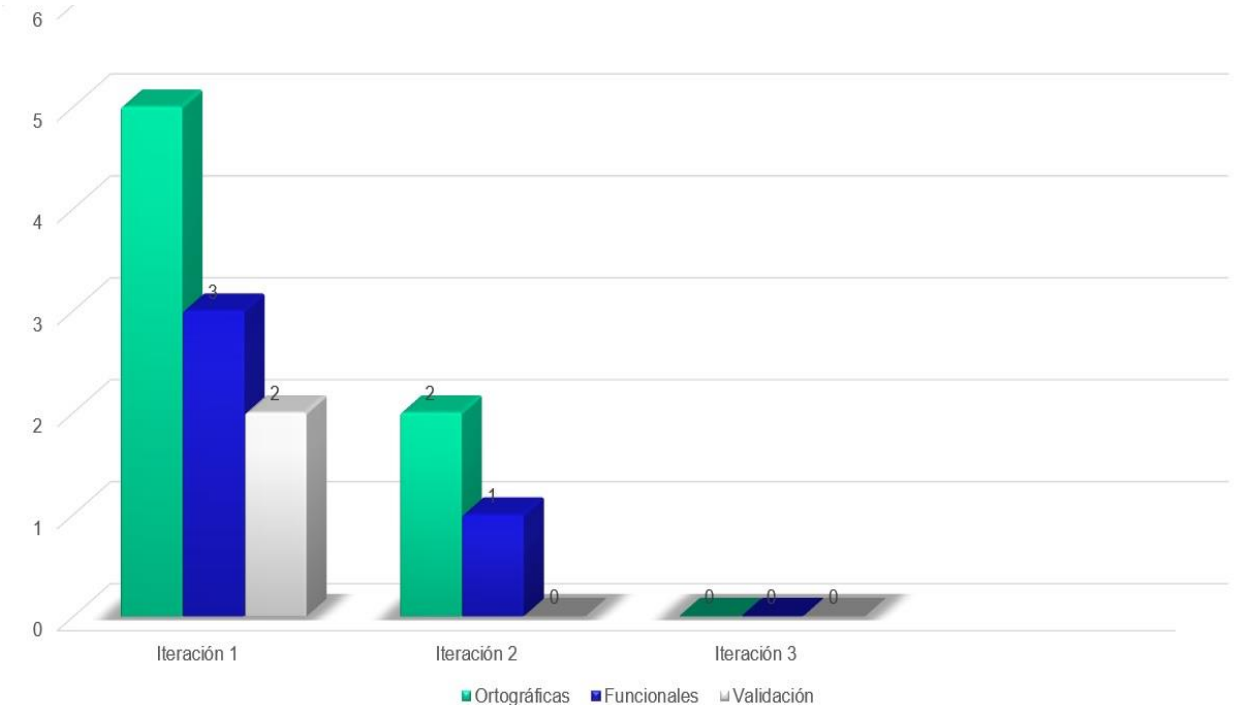

Fig. 20- Gráfica de las no conformidades encontradas durante las pruebas

3.5. Conclusiones del Capítulo

En el presente capítulo se realizó el modelo de implementación del *bundle Qooxdoo-Symfony* 2. Como parte de este se diseñaron los diagramas de componentes para mostrar una representación de los componentes físico del sistema. Se definieron los estándares de codificación a utilizar para lograr un código fuente más organizado y entendible, además de facilitar su posterior mantenimiento. Se le realizaron pruebas al sistema a nivel de desarrollador, a nivel de integración y a nivel de aceptación para garantizar un producto sin errores y con un buen funcionamiento. Para esto se utilizó, el método de caja blanca con la técnica de camino básico y pruebas unitarias, dentro del nivel de prueba de integración se realizó utilizó el método de caja negra se aplicó la técnica de partición de equivalencia y dentro del nivel de pruebas de aceptación se realizó el tipo de prueba Alfa. Se identificaron 13 no conformidades que fueron resueltas a medida que avanzaron las iteraciones.

CONCLUSIONES

Luego de haber terminado la presente investigación se arriban a las siguientes conclusiones:

- Se realizó un estudio de las principales características que presentaban algunos de los *bundles* de *framework JavaScript* existentes para *Symfony 2*, para a partir de estos escoger las funcionalidades necesarios con las que debía contar el *bundle Qooxdoo-Symfony2.*
- Con el desarrollo del *bundle QooxdooSymfony2* se le dio cumplimiento a 3 requisitos funcionales identificados durante la fase de análisis.
- A partir de las funcionalidades descritas se realizó el análisis y el modelo de diseño del *bundle Qooxdoo-Symfony2* proporcionando el punto de partida para las actividades de implementación.
- El diseño y ejecución de las pruebas a nivel de desarrollador, de integración y aceptación permitió comprobar el correcto funcionamiento del *bundle Qooxdoo-Symfony2.*
- Como resultado se obtuvo el *bundle Qooxdoo-Symfony2***,** el cual permite el desarrollo de aplicaciones web que utilicen los *frameworks JavaScript Qooxdoo y PHP Symfony 2.*

REFERENCIAS

CABRERA, R. F. A. C. S. A. A. *Metodologías Tradicionales Vs. Metodologías Ágiles*. Edtion ed., 2008.

DEVELOPED, Q. Qooxdoo Documentation. In., 2014.

DIENTUKI. Crear tu propio servidor web: 4. Apache, PHP y MySQL. 2008. Available from Internet:<http://www.cristalab.com/tutoriales/crear-tu-propio-servidor-web-4.-apache-php-y-mysql-c51133l/>.

ECLIPSE. OpenUp. 2012. Available from Internet:<http://epf.eclipse.org/wikis/openup/>.

EGUILUZ, J. *Introducción a Java Script*. Edtion ed., 2007.

EGUILUZ, J. *Desarrollo web ágil con Symfony2*. Edtion ed., 2014.

FELIX GARCÍA, Ó. R. A. C. S. Escuela de Ingeniería Civil Informática [online]. 2010. Available from World Wide Web:<http://www.eici.ucm.cl/Academicos/R_Villarroel/descargas/ing_sw_1/Metodologias.pdf>.

JACOBSON, G. B. A. J. R. A. I. El lenguaje Unificado de Modelado 2010.

JEFFRIES, R., ANDERSON, A., HENDRICKSON, C Extreme Programming Installed 2001.

JETBRAINS. PhpStorm. 2015. Available from Internet:<https://www.jetbrains.com/phpstorm/>.

LARMAN, C. UML y Patrones. In *Introduccion al analisis y diseno orientado a objetos.*, 2004.

MACIAS, G. MODELO N CAPAS. In., 2013.

OLEA, I. Integración de software [online]. 2014. Available from World Wide Web:<http://olea.org/servicios/integracion-software.html>.

PARADIGM, V. UML & SysML Toolset. In., 2012.

PENADÉS, P. L. Y. C. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. edited by U.P.D. VALENCIA. Edtion ed., 2011.

PREESMAN, R. S. *Inginiería de Software. Un enfoque práctico 6ta edición*. Edtion ed., 2011.

RUIZ, P. L. A. F. Lenguaje Unificado de Modelado - UML [online]. 2010. Available from World Wide Web:<http://ocw.unican.es/ensenanzas-tecnicas/ingenieria-del-software-i/materiales-de-clase-1/is1-t02-trans.pdf>.

SCHWABER, K., MIKE BEEDLE *Agile Software Development with Scrum,*. edited by P. HALL. Edtion ed., 2001.

SENCHA. Licencias de Sencha. 2015, [cited 10 de marzo del 2015. Available from Internet:<http://www.sencha.com/legal/>.

SOA. Framework. 2010. Available from Internet:<http://soaagenda.com/journal/articulos/que-son-los-frameworks>.

SOMMERVILLE, I. *Software Engineering Sommerville 8va Edición*. Edtion ed., 2007.

THOMPSON, I. Promonegocios.net. In., 2009.

TORRES, M. A. PHP. 2008. Available from Internet:<http://www.programacionweb.net/articulos/articulo/?num=686/>.

WEAVER, F. P. Y. R. Symfony 2.4, el libro oficial [online]. 2014. Available from World Wide Web:<http://librosweb.es/symfony_2_4/>.

WESLEY, A. *Extreme Programming Explained. Embrace Change*. edited by P. EDUCATION. Edtion ed., 2000.

ZANINOTTO, F. P. Y. F. *Symfony la guía definitiva*. Edtion ed. París, 2008.

BIBLIOGRAFÍA

CABRERA, R. F. A. C. S. A. A. *Metodologías Tradicionales Vs. Metodologías Ágiles*. Edtion ed., 2008.
DEVELOPED, Q. Qooxdoo Documentation. In., 2014.
DIENTUKI. Crear tu propio servidor web: 4. Apache, PHP y MySQL. 2008. Available from Internet:<http://www.cristalab.com/tutoriales/crear-tu-propio-servidor-web-4.-apache-php-y-mysql-c51133l/>.
ECLIPSE. OpenUp. 2012. Available from Internet:<http://epf.eclipse.org/wikis/openup/>.
EGUILUZ, J. *Introducción a Java Script*. Edtion ed., 2007.
EGUILUZ, J. *Desarrollo web ágil con Symfony2*. Edtion ed., 2014.
JACOBSON, G. B. A. J. R. A. I. El lenguaje Unificado de Modelado 2010.
JEFFRIES, R., ANDERSON, A., HENDRICKSON, C Extreme Programming Installed 2001.
JETBRAINS. PhpStorm. 2015. Available from Internet:<https://www.jetbrains.com/phpstorm/>.
LARMAN, C. UML y Patrones. In *Introduccion al analisis y diseno orientado a objetos.*, 2004.
MACIAS, G. MODELO N CAPAS. In., 2013.
OLEA, I. Integración de software [online]. 2014. Available from World Wide Web:<http://olea.org/servicios/integracion-software.html>.
PARADIGM, V. UML & SysML Toolset. In., 2012.
PENADÉS, P. L. Y. C. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. edited by U.P.D. VALENCIA. Edtion ed., 2011.
PREESMAN, R. S. *Inginiería de Software. Un enfoque práctico 6ta edición*. Edtion ed., 2011.
RUIZ, P. L. A. F. Lenguaje Unificado de Modelado - UML [online]. 2010. Available from World Wide Web:<http://ocw.unican.es/ensenanzas-tecnicas/ingenieria-del-software-i/materiales-de-clase-1/is1-t02-trans.pdf>.
SANTOS, F. G. A. Ó. R. A. C. Escuela de Ingeniería Civil Informática [online]. 2010. Available from World Wide Web:<http://www.eici.ucm.cl/Academicos/R_Villarroel/descargas/ing_sw_1/Metodologias.pdf>.
SCHWABER, K., MIKE BEEDLE *Agile Software Development with Scrum,*. edited by P. HALL. Edtion ed., 2001.
SENCHA. Licencias de Sencha. 2015, [cited 10 de marzo del 2015. Available from Internet:<http://www.sencha.com/legal/>.
SOA. Framework. 2010. Available from Internet:<http://soaagenda.com/journal/articulos/que-son-los-frameworks>.
SOMMERVILLE, I. *Software Engineering Sommerville 8va Edición*. Edtion ed., 2007.
THOMPSON, I. Promonegocios.net. In., 2009.
TORRES, M. A. PHP. 2008. Available from Internet:<http://www.programacionweb.net/articulos/articulo/?num=686/>.
WEAVER, F. P. Y. R. Symfony 2.4, el libro oficial [online]. 2014. Available from World Wide Web:<http://librosweb.es/symfony_2_4/>.

ANEXOS

WESLEY, A. *Extreme Programming Explained. Embrace Change*. edited by P. EDUCATION. Edtion ed., 2000.
ZANINOTTO, F. P. Y. F. *Symfony la guía definitiva*. Edtion ed. París, 2008.

UCI Informáticas

CARTA DE ACEPTACIÓN

En cumplimiento con la fase de desarrollo y en función de la ejecución de la tesis "Integración de Qooxdoo y Symfony 2" se hace entrega de los productos que se relacionan a continuación:

- Integración de Qooxdoo y Symfony 2(codigo fuente)
- Manual de usuario

Entrega	Recibe
Nombre y apellidos:	**Nombre y apellidos**
Laritza López Lobaina Frank Yasel de Armas Mora	Glennis Tamayo Morales
Cargo: Tesistas	**Cargo:** Jefe de Departamento Desarrollo de Componentes
Firma:	**Firma:**

UCI Universidad de las Ciencias Informáticas
Decanato Facultad 6

Consola de Symfony 2 con el Caso de Uso Generar Bundle de Qooxdoo

```
Symfony > qooxdoo:generate:bundle

  Bienvenido al Generador de Bundles Qooxdoo de Symfony2


El codigo de su aplicacion debe estar escrito en bundles.
Cada bundle está alojado bajo un namespace(como Acme/Bundle/BlogBundle).
El namespace debe comenzar con un nombre de proveedor, el nombre de su empresa,
el nombre de su proyecto o el nombre de su cliente
(A continuación de este nombre debe estar el sufijo Bundle).

Bundle namespace: UCI/BBundle

En el código, un bundle es a menudo referenciado por su nombre
Para tener un nombre único(una buena práctica es comenzar con el nombre del proveedor).
Basado en su namespace recomendamos UCIBBundle.

Bundle name [UCIBBundle]: BBundle

El bundle puede ser generado en cualquier directorio, el directoorio predeterminado es:

Target directory [/var/www/html/qooxdoosymfony2.git/src]:

Formato usado para la configuración.

Formatos de configuración (yml, xml, php, or annotation): yml

Para agilizar el proceso el comando lo ayudara a generar los fragmentos de código por usted

¿Quieres generar toda la estructura de directorios? [no]

  Resumen después de la generación

Usted va a generar el "UCI\BBundle\BBundle" bundle
en "/var/www/html/qooxdoosymfony2.git/src/" usando el formato "yml".

¿Quieres confirmar la generación? [yes]

  Bundle de Qooxdoo Generado

El codigo del bundle se ha generado: OK
Comprobación de que el paquete se carga automáticamente: OK
¿Confirmar la actualización automática de su Kernel? [yes]
Paquete habilitado dentro del Kernel OK
¿Confirma la actualización automática de las rutas? [yes]
Importación de las rutas del bundle: OK
```

Consola de Symfony 2 con el Caso de Uso Generar Compilar Bundle de Qooxdoo

```
Symfony > qooxdoo:compile
Bienvenido al comando para compilar bundle de qooxdoo

Entre el namespace del bundle a compilar UCI/BBundle
Entre las opciones de compilacion, por defecto esta en null
Comienza el proceso de compilacion espere unos segundos

Este proceso puede tardar en dependencia de los
cambios hechos en el proyecto o si es compilado por primera vez

  - Warning: ! Shadowing job "libraries" with local one
El bundle se ha compilado: OK
Installing assets as hard copies
Installing assets for Symfony\Bundle\FrameworkBundle into web/bundles/framework
Installing assets for UCI\NuevoBundle into web/bundles/nuevo
Installing assets for UCI\ABundle into web/bundles/a
Installing assets for UCI\BBundle into web/bundles/b
Installing assets for Acme\DemoBundle into web/bundles/acmedemo
Installing assets for Sensio\Bundle\DistributionBundle into web/bundles/sensiodistribution
Symfony >
```

Consola de Symfony 2 con el Caso de Uso Generar CRUD de una entidad a un Bundle de Qooxdoo