Universidad de las Ciencias Informáticas

Facultad 3

Título:

Interfaz gráfica para la configuración de los componentes Caché, Excepciones, IUX, Aspectos y Trazas del marco de trabajo Bosón.

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

Autor(es): Daniel Herrera Sánchez

Tutor(es): Ing. René Rodrigo Bauta Camejo

Ing. Claudia Bravo Batista

La Habana. Junio del 2016

Declaración de autoría

Declaro ser autor de la presente tesis que tiene por título: Interfaz gráfica de usuario para la configuración de los componentes Caché, Excepciones, IUX, Aspectos y Trazas del marco de trabajo Bosón, y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Daniel Herrera Sánchez **Ing. René Rodrigo Bauta Camejo**

_____ _____

Firma del Autor Firma del Tutor

Ing. Claudia Bravo Batista

_____

Firma del Tutor

Datos de Contacto

René Rodrigo Bauta Camejo: Ingeniero en Ciencias Informáticas.

Claudia Bravo Batista: Ingeniero en Ciencias Informáticas, graduado en el 2012.

Dedicatoria

A mis padres y a mi abuela Margarita, por haber estado a mi lado y por apoyarme en todo momento de este trayecto.

Agradecimientos

A mis padres y mi abuela Margarita por el apoyo constante y los ánimos para llegar hasta donde estoy hoy.

A mis tíos Ramón y Maritza por estar siempre pendientes de mí y dispuestos a ayudarme en todo lo que ha hecho falta.

A mis amigos Diego, Yotsan, Pavel, Dayán, David, José Carlos, Andrés, Carlos, Lientz, Eimé …. por hacer de estos 5 años en la universidad una experiencia inolvidable.

A mi tutora Claudia, por todo el apoyo brindado en la realización de este trabajo y tantas correcciones.

Resumen

En los últimos tiempos la comunidad de desarrollo de software se ha estado enfocando en la creación y utilización de marcos de trabajo para la construcción de sistemas de información basados en la web. El marco de trabajo Bosón ha sido creado en la Universidad de las Ciencias Informáticas con el fin de agilizar el proceso de desarrollo de aplicaciones proponiendo un conjunto de buenas prácticas, tecnologías y componentes que responden a la mayoría de los requisitos tecnológicos de un amplio espectro de aplicaciones. Estos componentes tienen bajos índices de usabilidad en cuanto a la configuración de los mismos. Esto se debe a que para configurar los componentes es necesario realizar una búsqueda de los ficheros de configuración dentro de la aplicación, y luego en dependencia del formato que tengan deben ser modificados. Con la solución propuesta se brinda un conjunto de interfaces gráficas de usuario desarrolladas sobre tecnologías libres que facilitan este proceso de búsqueda y modificación a los desarrolladores de software. Estas interfaces cumplen con los indicadores de usabilidad propuestos por la ISO 9241-11 y suponen una mejora considerable sobre los procedimientos anteriores.

Palabras clave: Bosón, componente, configuración, interfaz, usabilidad

Abstract

Nowadays software developer's community has been focusing on the creation of frameworks in order to build web-based Information Systems. Bosón is a framework that was created in the University of Informatics Sciences with the purpose of accelerate the application developing process by implementing good practices, technologies and components already built and waiting to be used. These components have a very low usability index as far as configuration concerns. This happens because to configure the components it's necessary to perform a search among the configuration files within the application, and then regarding the format of the file make the changes. The main intention of the solution is to facilitate this task to programmers. These interfaces meet the usability indicators given in the ISO 9241-11 and they represent a considerable improvement over the old procedures.

Keywords: Bosón, component, configuration, interface, usability

Índice

Índice de figuras

Introducción

Hasta hace algunos años los ordenadores eran máquinas de gran tamaño, poco comunes y estaban equipados con interfaces de líneas de comando (CLI por sus siglas en inglés) como vía de interacción con el usuario. Con el tiempo la tecnología fue evolucionando y el mercado con ella; se fueron incorporando ordenadores cada vez más pequeños a las grandes empresas junto con las innovaciones que se producían. Una de ellas fueron las interfaces gráficas de usuario (GUI por sus siglas en inglés), que supusieron una mejora abismal con respecto a las CLI en el hecho de que hacían las operaciones con ordenadores más intuitivas.

Es un hecho comprobado científicamente que el cerebro trabaja de forma más eficiente con iconos gráficos que con palabras. Las palabras añaden una capa extra de interpretación al proceso de comunicación. Además de lo intuitivo de las operaciones está el hecho de que las interfaces gráficas de usuario generalmente devuelven inmediatamente una reacción o efecto ante cada acción. Por ejemplo, cuando un usuario borra un icono que representa un archivo el icono inmediatamente desaparece confirmando que el archivo ha sido borrado, lo cual no ocurre en las CLI. Actualmente las interfaces gráficas de usuario se han convertido en un estándar para la interacción hombre-máquina, y han conducido al desarrollo de nuevos tipos de aplicaciones. (Shneiderman, 1998)

En el Centro de Informatización Entidades (CEIGE) de la Universidad de las Ciencias Informáticas se desarrolla el marco de trabajo Bosón. El mismo se ha creado con el objetivo de agilizar el proceso de desarrollo de aplicaciones, proponiendo un conjunto de buenas prácticas, patrones, tecnologías y componentes implementados sobre Symfony 2. Estos componentes responden a la mayoría de los requisitos tecnológicos de un amplio espectro de aplicaciones.

En Bosón el proceso de configuración de los componentes Caché, IUX, Excepciones, Aspectos y Trazas se hace a través de la modificación de archivos de configuración o a través de comandos en consola. Por ejemplo, para borrar la caché de la aplicación, es necesario localizar en el servidor la carpeta encargada de su almacenamiento y eliminarla. Para la gestión de excepciones, es necesario acceder a los archivos del componente, buscar el encargado de almacenar las mismas y modificarlo respetando siempre su estructura. Estos archivos de configuración para los componentes pueden ser encontrados en diversas localizaciones dentro de la aplicación, y no siempre están en el mismo formato.

De manera similar ocurre con el resto de los componentes, implicando que solo los usuarios con dominio avanzado en informática puedan acceder y gestionar la

información y que las operaciones no se puedan realizar de forma sencilla producto a que se deben memorizar un conjunto de comandos sobre las CLI. Esto afecta en gran medida el grado de usabilidad de los componentes Excepciones, IUX, Aspectos, Trazas y Caché en Bosón, limitando su uso a un ámbito muy pequeño de usuarios para cumplir sus objetivos específicos con eficacia y eficiencia.
Luego de analizada la problemática existente se identifica como **problema a resolver**: ¿Cómo mejorar la usabilidad de los componentes Excepciones, IUX, Aspectos, Trazas y Caché del marco de trabajo Bosón?

Se plantea como **objeto de estudio** el proceso de desarrollo de interfaces gráficas de usuarios en las aplicaciones web, centrado en su **campo de acción**: Proceso de desarrollo de interfaces gráficas de usuarios en los componentes de Caché, Excepciones, IUX, Aspectos y Trazas.

Para darle solución al problema de investigación se plantea el **objetivo general**: Desarrollar la interfaz gráfica de usuario para la configuración de los componentes Caché, Aspectos, Excepciones, IUX y Trazas del marco de trabajo Bosón y se definen los **objetivos específicos:**

- Fundamentar la selección de la metodología, herramientas y las tecnologías a utilizar en el desarrollo de las interfaces gráficas de usuarios.
- Analizar y diseñar la propuesta de desarrollo de las interfaces gráficas de usuario de los componentes Caché, Excepciones, IUX, Aspectos y Trazas en el marco de trabajo Bosón.
- Implementar la propuesta de desarrollo de las interfaces gráficas de usuarios de los componentes Caché, Excepciones, IUX, Aspectos y Trazas en el marco de trabajo Bosón.
- Validar la investigación y la solución propuesta.

Para cumplir con los objetivos se desglosan las siguientes **tareas de investigación**:

- Confección del marco teórico de la investigación a partir de la búsqueda y revisión bibliográfica de las tecnologías que permiten construir interfaces gráficas para las aplicaciones web.
- Análisis de la estructura y funcionamiento de los componentes Caché, Excepciones, IUX, Aspectos y Trazas del marco de trabajo Bosón.
- Elaboración del modelo de diseño de las interfaces gráficas de los componentes Caché, Excepciones, Aspectos, IUX y Trazas del marco de trabajo Bosón.
- Implementación de las interfaces gráficas de los componentes Caché, Excepciones, IUX, Aspectos y Trazas del marco de trabajo Bosón.

- Validación de la solución propuesta mediante las Pruebas de Caja Negra e Integración.
- Validación de la investigación mediante un pre-experimento.

Se propone como **idea a defender** de la investigación: Si se desarrolla la interfaz gráfica de usuario para la configuración de los componentes Caché, Excepciones, Aspectos, IUX y Trazas del marco de trabajo Bosón se mejora la usabilidad de los mismos.

Para llevar a cabo la investigación se utilizan los siguientes métodos científicos detallados a continuación:

Métodos teóricos:

Se utiliza el método **histórico-lógico** para realizar un análisis de los cambios de paradigmas en la estructura de los marcos de trabajos: Phalcon, Zend Framework, AngularJS, Dojo Toolkit entre otros, que permitan obtener información para la construcción de las interfaces gráficas en Bosón.

Se utiliza el método **analítico-sintético** para separar en partes el proceso de desarrollo de las interfaces gráficas de usuarios de los componentes Caché, Excepciones, IUX, Aspectos y Trazas que permitan ejecutar las tareas que respondan al cumplimiento del objetivo general.

A través de la **modelación** se realiza un diseño del producto final que se va a obtener, con características específicas.

Métodos empíricos:

El método empírico utilizado fue el **análisis estático** y se aplica a los componentes en cuestión para obtener información sobre la estructura y funcionamiento de los mismos, a través de la revisión del código fuente. Según (Barchini, 2005) el análisis estático es un método observacional que recoge los datos relevantes a medida que se va desarrollando el proyecto.

El presente documento está conformado por tres capítulos estructurados de la siguiente manera:

Capítulo I: Fundamentación teórica

En este capítulo se hace referencia a los elementos teóricos que soportan la investigación, se presentan los lenguajes de programación, así como las tecnologías y metodologías que se ajustan al desarrollo del trabajo.

Capítulo II: Análisis y diseño de la solución propuesta.

En este capítulo se realiza la descripción del sistema, se identifica y describen los requisitos funcionales y no funcionales, se diseñan las interfaces gráficas de usuarios mediante los productos de trabajo diagrama de clases del diseño, representando las

clases de la implementación y las relaciones entre las mismas. Además, se define la arquitectura de la aplicación mediante los estilos y patrones arquitectónicos.

Capítulo III: Implementación, prueba y validación.

En este capítulo se abordan los elementos de estándares de codificación y la implementación, utilizando el lenguaje de programación y la metodología seleccionada. También son presentados los resultados de aplicar pruebas de caja negra al sistema utilizando la técnica de partición equivalente y prueba de integración descendente definida por (Pressman, 2010). Se valida la investigación realizada mediante un pre-experimento.

Capítulo 1: Fundamentación teórica

Introducción

El objetivo de este capítulo es desarrollar la fundamentación teórica, la cual incluye el estudio de los principales conceptos a tener en cuenta en la investigación. Además, se presentan la metodología a emplear, los principales lenguajes de programación y entornos de desarrollo, así como algunas características de los mismos.

1.1 Conceptos asociados al dominio del problema

En el siguiente sub-epígrafe se establece una línea base en la investigación en cuanto a conceptos básicos tratados y componentes involucrados.

1.1.1 Componente

Sobre los componentes varios autores proponen diferentes definiciones:
“Los componentes son unidades de software independientes que pueden estar compuestas por otros componentes y que se utilizan para crear un sistema software”. (Sommerville, 2005)
“Un elemento de software que se ajusta a un modelo de componentes y que puede ser desplegado y compuesto de forma independiente sin modificación, según un estándar de composición”. (Heineman, et al., 2001)
“Un componente es una unidad de composición con interfaces especificadas contractualmente y dependencias de contexto explícitas únicamente. Un componente software puede ser desplegado de forma independiente y está sujeto a la composición por terceras partes”. (Szyperski, 1998)
Luego de analizar algunos de los conceptos de componente, es posible llegar a la siguiente conclusión: "Un componente es una unidad independiente básica de composición de un sistema que define interfaces y dependencias".
Según (Calás Torres, 2014) el marco de trabajo Bosón materializa los requisitos comunes de las arquitecturas base para sistemas de gestión web a partir del desarrollo de los siguientes componentes:
Aspectos: Emplea la Arquitectura Orientada a Aspectos mediante la definición de reglas de negocio que pueden ser ejecutadas antes o después de las acciones de un controlador en un orden definido. Los parámetros de aspectos son almacenados en el fichero *aspects.yml,* localizado en la carpeta de recursos de cada componente. Este fichero tiene como parámetros el nombre del aspecto, el controlador y la acción a ejecutarse, el orden de ejecución, entre otros.

Caché: Facilita a las aplicaciones que se están desarrollando, gestionar la caché de forma rápida y sencilla, permite guardar información en cualquiera de los formatos: texto plano, arreglos o tablas hash, objetos o instancias de clases, estructuras complejas como listas, pilas, colas, árboles y otras estructuras de mayor complejidad. El fichero que contiene la configuración de la caché puede tener cualquier nombre, aunque siempre debe estar referenciado desde el fichero *config.yml* y contener los parámetros*:* anfitrión, puerto y URL.

Excepciones: Permite controlar de forma centralizada los diferentes tipos de excepciones y tratarlas según el tipo especificado. Permite además la internacionalización de los mensajes de las excepciones. Las excepciones se encuentran almacenadas en la carpeta de recursos de cada componente, específicamente en el fichero *excepciones.yml*. Las traducciones por su parte se encuentran en la carpeta *translateexceptions*. Los principales parámetros de las excepciones son: código, mensaje y descripción.

IUX: Permite la creación de interfaces utilizando el proyecto de Interfaz Única (IUX), el cual define estándares de diseño para cada una de las líneas temáticas de la Universidad de las Ciencias Informáticas. Integra el IUX con el motor de plantillas utilizado por Symfony2 para el desarrollo de interfaces. Proporciona a los desarrolladores la posibilidad de elegir qué línea desea utilizar, para así generar automáticamente elementos gráficos ajustados a la línea temática. El fichero que contiene la configuración de IUX puede tener cualquier nombre, aunque siempre debe estar referenciado desde el fichero *config.yml* y contener los parámetros*:* línea temática, marco, nombre del producto, eslogan, entre otros.

Trazas: Detecta y registra las trazas generadas por un sistema a partir de la captura de eventos por lo cual resulta útil para las auditorias. Permite registrar trazas de tipo acción, rendimiento, acceso a datos y excepciones. El fichero que contiene la configuración para activar o desactivar las trazas puede tener cualquier nombre, aunque siempre debe estar referenciado desde el fichero *config.yml* y contener los parámetros*:* datos, excepciones, rendimiento, acción.

1.1.2 Interfaz Gráfica de Usuario

El objetivo general de este trabajo está orientado al desarrollo de interfaces gráficas para la configuración de los componentes Caché, Aspectos, Excepciones, IUX y Trazas del marco de trabajo Bosón. Sobre este término algunos autores han emitido diferentes conceptos, siendo algunos de estos:

“La interfaz gráfica de usuario es un programa informático que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador”. (Royo, 2004)

“En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático”. (Shneiderman, 1998)

A partir de los conceptos analizados, se concluye que la interfaz gráfica de usuario es un medio que permite la interacción entre seres humanos y ordenadores a través de imágenes, gráficos y acciones disponibles.

1.1.3 Usabilidad

El término de usabilidad es de vital importancia para los entornos web, dado que los mismos se están convirtiendo en un elemento clave para el desarrollo de las empresas las cuales ofertan gran cantidad de información y servicios. Sobre este término algunos autores han dado diferentes conceptos.

"La usabilidad de un sistema es la capacidad humana en términos funcionales, de que el mismo pueda utilizarse con facilidad y eficacia por el rango especificado de usuarios y recibir una formación específica de apoyo, para cumplir con el rango especificado de tareas, dentro del rango especificado de escenarios." (Shackel, 1991)

De acuerdo con la norma ISO 9241-11 la usabilidad es el grado en que un producto puede ser utilizado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un determinado contexto.

Esta norma ISO establece que para que un sistema sea usable debe cumplir con los siguientes indicadores:

- **Fácil de aprender**: Hace referencia a cuán fácil es para nuevos usuarios aprender a realizar las operaciones sobre la aplicación.
- **Fácil de recordar**: Hace referencia a cómo usuarios intermitentes sean capaces de recordar los procedimientos para interactuar con la aplicación.
- **Efectivo**: Hace referencia al grado en el que la aplicación cumpla con las necesidades reales del usuario, y le sea de utilidad para realizar las tareas que se proponga.

- **Eficiente**: Hace referencia al número de transacciones por unidad de tiempo que el usuario sea capaz de realizar. Cuanto más usable es el sistema, más rápido es el usuario interactuando con el mismo.
- **Baja Tasa de Errores**: Hace referencia al número de errores que puede cometer un usuario mientras realiza una operación sobre el sistema. Mientras mayor es la cantidad de errores menos eficiente y efectiva es la aplicación, así como menor es la satisfacción del usuario.

1.2 Análisis de soluciones existentes

En el presente sub-epígrafe se analizan marcos de trabajo para la construcción de aplicaciones web que tienen implementados los componentes Caché, Excepciones, Aspectos, IUX y Trazas, con el objetivo de realizar el diseño de las interfaces gráficas de usuarios en los componentes de Bosón.

Los marcos de trabajo a analizar son Sauxe, Yii, Zend y Phalcon debido a que siguen la filosofía planteada por la ingeniería de software basada en componentes teniendo implementados modelos de componentes, y aceptan la integración de otras tecnologías para la creación de interfaces gráficas de usuarios. De acuerdo a (Yii Framework, 2016) (Zend Technologies Ltd, 2015) (PhalconPHP, 2016) los marcos de trabajo seleccionados se comportan de la siguiente manera:

1.2.1 Componente Caché

Marcos de trabajo	Descripción
Sauxe	No está implementado un componente para la gestión de la caché. Si se desea por ejemplo eliminar la caché de la aplicación es necesario acceder a la carpeta Caché del servidor de la aplicación y eliminarla manualmente.
Zend	No cuenta con una interfaz gráfica para la gestión de la caché. Se utilizan las clases localizadas en Zend_Cache. El uso de la caché se hace a través de los llamados Frontends y son almacenados utilizando adaptadores de Backends (archivos, bases de datos, entre otros.
Yii	No cuenta con una interfaz gráfica para la gestión de la caché. Se utilizan las clases: CMemCache, CApcCache, CDbCache.
Phalcon	No cuenta con una interfaz gráfica para la gestión de la caché. Se utilizan las clases: Phalcon\Cache, Phalcon\Cache\Frontend, Phalcon\Cache\Backend.

1.2.2 Componente Trazas

Marco de trabajo	Descripción
Sauxe	Tiene implementado un componente para la gestión de las trazas. Las interfaces visuales fueron desarrolladas utilizando la librería de JavaScript ExtJS 2.2. En el anexo 1 se muestran dos capturas de la interfaz gráfica de este componente. El marco de trabajo brinda la posibilidad de activar o desactivar las trazas, así como obtener un reporte del registro de trazas de la aplicación.
Zend	No está implementado un componente para la gestión de las trazas. Se emplea el conjunto de clases localizadas en Zend\Log.
Yii	No está implementado un componente para la gestión de las trazas. Se emplean las clases: CLogRouter, CFileLogRoute, CEmailLogRoute.
Phalcon	No está implementado un componente para la gestión de las trazas. Se emplea el conjunto de clases localizadas en Phalcon\Logger.

1.2.3 Componente Excepciones

Marco de trabajo	Descripción
Sauxe	No está implementado un componente para la gestión de las excepciones. Para adicionar, editar o eliminar excepciones se hace necesario localizar el fichero exceptions.xml y editarlo respetando su formato previamente establecido.
Zend	No está implementado un componente para la gestión de las excepciones. Se utiliza la clase Zend_Exception, la cual hereda de la clase Exception de PHP. Esta clase se puede utilizar para capturar todas las excepciones lanzadas por el marco de trabajo.
Yii	No está implementado un componente para la gestión de las excepciones. Se utilizan las clases: CException, CDbException, CHttpException.
Phalcon	No está implementado un componente para la gestión de las excepciones. Se utiliza la clase Phalcon\Exception.

1.2.4 Componente IUX

Este componente no está implementado en ninguno de los marcos de trabajo analizados.

1.2.5 Componente Aspectos

Marco de trabajo	Descripción
Sauxe	Tiene implementado un componente para la gestión de los aspectos. Las interfaces visuales fueron desarrolladas utilizando la librería de JavaScript ExtJS 2.2. En el anexo 2 se muestra una captura de la interfaz gráfica de este componente.

1.3 Metodología de desarrollo

En el documento de la arquitectura vista de desarrollo tecnológico del marco de trabajo Bosón, se define como metodología a emplear una variación de AUP (Agile Unified Process, por sus siglas en inglés) apoyándose en el modelo CMMI-DEV V1.3, que se adapte a la actividad productiva de la Universidad de las Ciencias Informáticas. Esta metodología es conocida como Metodología de Desarrollo para la Actividad Productiva de la UCI.

La metodología propuesta por (Rodríguez Sánchez, 2014) describe tres fases, las cuales son: Inicio, Ejecución y Cierre. La investigación se centra en la fase de Ejecución, en la cual se ejecutan las actividades requeridas para el desarrollo del software, se obtienen los requisitos, se elabora la arquitectura y el diseño, se implementa y se libera el producto. Las disciplinas que se emplearán son: Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de aceptación. El expediente de proyecto a utilizar para almacenar los artefactos generados o productos de trabajo se encuentra disponible en la dirección http://excriba.prod.uci.cu. Se hará uso del **Escenario 4** de la metodología, por lo cual el método de encapsulación de requisitos es a través de **Historias de Usuario**.

1.4 Entorno de desarrollo

1.4.1 Lenguaje Unificado de Modelado

De acuerdo a (Rumbaugh, et al., 1999) UML es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema orientado a objetos. Dentro de los principales beneficios de utilizar UML se encuentran:

- Mejores tiempos totales de desarrollo (de 50% o más).
- Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- Es un lenguaje de modelado utilizado tanto por humanos como por máquinas.

- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización y minimización de costos.

1.4.2 Herramientas CASE

Las herramientas para la Ingeniería de Software Asistida por Computadora (CASE, por sus siglas en inglés) intervienen en casi todos los aspectos del ciclo de vida de desarrollo del software, específicamente en tareas tales como el diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente, compilación automática, documentación o detección de errores, entre otras. Estas herramientas están destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de capital. (Sitio web oficial del producto Visual Paradigm, 2014)

Desde que se crearon estas herramientas en 1984 hasta la actualidad, cuentan con una credibilidad y exactitud que tienen un reconocimiento universal, siendo usadas por un desarrollador que busca un resultado óptimo y eficiente, pero sobre todo que busca esa minuciosidad necesaria de los procesos y entre los procesos. (Instituto Nacional de Estadística e Informática, 1999)

1.4.3 Lenguajes de Desarrollo

Lenguaje de Marcado de Hipertexto (HTML)

Lenguaje utilizado para la publicación de hipertexto en la web y desarrollado con la idea de una persona o tipo de dispositivo pueda acceder a la información en la web. Es un lenguaje estático constituido por elementos que el navegador interpreta y los despliega en la pantalla de acuerdo con su objetivo. Se caracteriza por ser sencillo, presentar el texto de forma estructurada y agradable, no necesita de grandes conocimientos cuando se cuenta con un editor de páginas web, archivos pequeños, despliegue rápido, lenguaje de fácil aprendizaje y lo admiten todos los exploradores. (Lapuente, et al., 2008)

Hojas de Estilo en Cascadas (CSS)

Mecanismo que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en el documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre el estilo y formato de sus documentos.

Es utilizada para separar el contenido de la presentación y está basado en una serie de reglas, que rigen el estilo de los elementos en los documentos estructurados formando de esta forma la sintaxis de las hojas de estilo. Una de las principales características de CSS es su flexibilidad y las diferentes opciones que ofrece para realizar una misma

tarea. Usar CSS tiene varias ventajas porque posibilita un mayor control de la presentación agilizando de esta manera su actualización. La hoja de estilo se almacena en caché después de la primera solicitud y se puede volver a usar para cada página del sitio, de esta forma, contribuye con menos costes de almacenamiento y más velocidad a la hora de cargar las páginas web. (Briggs, 2003)

JavaScript

Es un lenguaje de programación ligero e interpretado. Los programas hechos en JavaScript se pueden probar en cualquier navegador sin necesidad de procesos intermedios. Permite el desarrollo de una interfaz de usuario mejorada, aumentando la experiencia del usuario en la web. (Chapman, 2007)

AngularJS

AngularJS es un marco de trabajo para aplicaciones web dinámicas. Permite la utilización de HTML y extender su sintaxis para crear los componentes de la aplicación de manera clara, objetiva y breve. Es enlazado a datos (proceso que conecta la interfaz de usuario con la lógica de negocio) y cuenta con la inyección de dependencias, la cual elimina gran parte del código que se debe escribir. Fue creado en el año 2009 por Misko Hevery y Adam Abrons, aunque hoy en día es mantenido por Google. Es un marco de trabajo para construir aplicaciones a gran escala, que requieran de alto rendimiento y a la vez sean sencillas de mantener. (AngularJS, 2015)

PHP

Lenguaje diseñado especialmente para desarrollo de aplicaciones web, puede ser incrustado dentro del código HTML y desplegado en casi todos los sistemas operativos, plataformas y en la mayoría de los servidores web. Como principales ventajas se puede mencionar la 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. Permite el manejo de excepciones, cuenta con una amplia documentación y soporte de la comunidad. (Rojas Morales, et al., 2004)

Symfony 2

Symfony 2 es un marco de trabajo para desarrollar aplicaciones PHP. Se anunció por primera vez a principios de 2009. (Eguiluz, 2013) Symfony 2 ha sido ideado para las aprovechar al máximo las nuevas características de PHP y es uno de los marcos de trabajo con mejor rendimiento. Su arquitectura interna está completamente desacoplada, lo que permite reemplazar o eliminar fácilmente aquellas partes que no encajan en el proyecto. (Eguiluz, 2013)

Principales características

- Su código, y el de todos los componentes y librerías que incluye, se publican bajo la licencia MIT de software libre.
- La documentación del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos.
- Cuenta con una gran comunidad que brinda el soporte necesario para desarrollar las aplicaciones.
- Incluye varias herramientas gráficas y de consola para depurar fácilmente los errores que se produzcan en las aplicaciones.
- Para evitar el uso de contraseñas en archivos de configuración, permite establecer los parámetros de configuración de las aplicaciones a través de variables de entorno del propio servidor.
- Utiliza la herramienta Composer para simplificar la instalación y gestión de las dependencias de las aplicaciones PHP.
- Dispone de un plan de lanzamientos predecible, con versiones estables mantenidas durante tres años.

El marco de trabajo Bosón está basado en Symfony para construir sus componentes. Las interfaces gráficas de usuario por su parte se desarrollan con AngularJS.

1.4.4 Entorno de Desarrollo Integrado

PHPStorm y WebStorm son entornos de desarrollo integrados (IDE por sus siglas en inglés) desarrollados por Jetbrains que proveen excelente soporte para PHP (incluyendo las últimas versiones del lenguaje y marcos de trabajo), HTML, JavaScript, CSS, Sass, Less, CoffeeScript, Node.js, AngularJS, entre otros. Tienen dentro de sus características la detección de errores, inspecciones y correcciones de código al vuelo. Poseen además funciones de búsquedas rápidas para saltar a cualquier parte de una clase, archivo, símbolo, o incluso acción del entorno en el IDE.

Tienen un depurador de código que permite conocer lo que está sucediendo en cada uno de los pasos que sigue la aplicación. Se integran además con depuradores como Xdebug y ZendDebugger y permiten realizar pruebas unitarias con PHPUnit. Todos los marcos de trabajo más importantes que utilizan PHP están soportados por PHPStorm, tanto como por el mismo IDE, como adicionando un complemento libre de costo. Se integran además con sistemas de repositorios como GIT, SVN, Mercurial, entre otros y permiten configurar en pocos pasos sistemas de despliegue vía FTP/SFTP/FTPS. (JetBrains SRO, 2015)

1.4.5 Sistemas de Bases de Datos

El sistema de bases de datos a utilizar es MySQL, en su versión 5.6.12. Está escrito en C y C++, y destaca por su gran adaptación a diferentes entornos de desarrollo,

permitiendo su interacción con los lenguajes de programación más utilizados como PHP, Perl y Java, y su integración en distintos sistemas operativos. Su principal objetivo es la velocidad y robustez. MySQL sigue políticas de código abierto, lo cual permite que su utilización sea gratuita e incluso se pueda modificar con total libertad, pudiendo descargar su código fuente. Esto ha favorecido su desarrollo y continuas actualizaciones, para hacer de MySQL una de las herramientas más utilizadas. (Sitio web oficial MySQL, 2015)

1.4.6 Servidor Web

El servidor a utilizar será Apache en su versión 2.4.4. Apache es el servidor web de mayor uso a nivel internacional. Apache es flexible, rápido y eficiente, además de ser continuamente actualizado. Entre sus características se destaca que es multiplataforma y puede ser adaptado a diferentes entornos y necesidades. Resaltar también que presenta un diseño modular excelente para ampliar las capacidades del servidor web, contando además con una amplia variedad de módulos que se pueden instalar a medida que se necesiten. (Sitio web oficial de Apache, 2015)

1.4.4 Estilos y patrones arquitectónicos.

Patrón Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador (MVC por sus siglas) es un patrón arquitectónico para el desarrollo de aplicaciones, el cual está compuesto por tres partes fundamentales:

- Modelo: Es el nivel más bajo de este patrón y es el responsable de mantener los datos.
- Vista: Es responsable de mostrar o bien todo o una porción de los datos al usuario.
- Controlador: Controla las interacciones entre el Modelo y la Vista.

MVC separa la lógica de la aplicación de la capa de interfaz de usuario. El controlador recibe todas las peticiones de la aplicación y entonces trabaja con el modelo para preparar los datos que necesita la vista. La vista usa los datos preparados por el controlador para generar una respuesta final. (Sommerville, 2005) La abstracción del modelo MVC puede ser gráficamente representada de la siguiente forma:

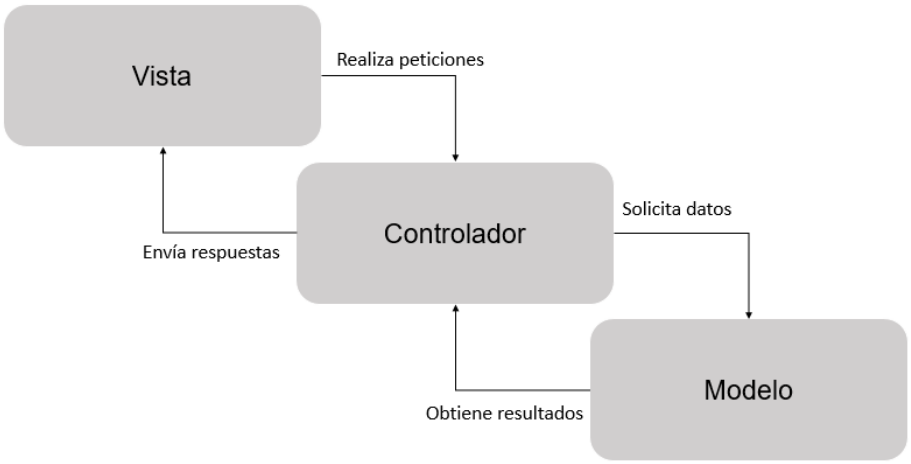


Figura 1. Representación del Patrón Modelo Vista Controlador

1.5 Métricas de validación del diseño

De acuerdo a (Pressman, 2010) medir la calidad del software es una tarea propensa al debate pues no existen métodos absolutos para lograrlo. Por su parte (McCall, et al., 1977), (Grady, et al., 1987) y (Card, et al., 1990) proponen un conjunto de listas de chequeo con el objetivo de medir la calidad de los atributos de software como por ejemplo: facilidad de mantenimiento, facilidad de prueba, reusabilidad, entre otros. Con respecto al diseño a nivel de componentes algunos autores han propuesto con similar objetivo un conjunto de métricas centrándose en las características internas de los componentes de software. (Bieman, et al., 1994) propone procedimientos para medir específicamente la cohesión entre los elementos de los componentes. (Dhama, 1995) y (McCabe, et al., 1994) proponen procedimientos para medir el acoplamiento y la complejidad como atributos de calidad del software.

El diseño de la solución propuesta se valida utilizando las métricas Tamaño operacional de clases (TOC por sus siglas) y Relaciones entre clases (RC por sus siglas). Según (DesarrolloWeb, 2015) estas son métricas de validación del diseño a nivel de componentes y evalúan gran parte de los atributos de software que se han mencionado, siendo esta la razón por la cual han sido seleccionadas. Los atributos medidos son:

- **Responsabilidad**. Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación**. Consiste en el grado de dificultad que tiene implementado un diseño de clases determinado.
- **Reutilización**. Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

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

1.6 Conclusiones del capítulo

Luego de haberle dado cumplimientos a los objetivos trazados para el primer capítulo, se arribaron a las siguientes conclusiones:

- De los marcos de trabajo analizados solamente presenta una interfaz gráfica de usuario para la configuración de los componentes el marco de trabajo Sauxe. El resto lo hace a través de clases o métodos.
- El análisis de la estructura y funcionamiento de los componentes permitió extraer información sobre los parámetros y procedimientos de configuración, así como la localización de los datos.
- La revisión de las herramientas, tecnologías y la metodología propuestas por el departamento Desarrollo de Componentes para el desarrollo de las interfaces de usuarios, permitió identificar las utilidades generales que brindan.
- El análisis de diferentes métricas y procedimientos para medir la calidad de atributos de software permitió seleccionar TOC y RC como métricas de validación del diseño, pues estas abarcan gran parte de los atributos de calidad de software.

Capítulo 2: Análisis y diseño de la aplicación propuesta

Introducción

El análisis tiene como propósito fundamental conseguir una comprensión más precisa de los requisitos y una descripción de los mismos. El diseño es más que analizar los requerimientos refinándolos y estructurándolos, permite dar forma al sistema de manera que proporcione vida a todos los requisitos (Rumbaugh, et al., 1999). En este capítulo se describen los requisitos funcionales los cuales deben satisfacer la aplicación. Además, se especifican las técnicas utilizadas para la captura y validación de los mismos. Son definidos los estilos y patrones arquitectónicos empleados en la solución. Por último, se presenta la validación del diseño propuesto.

2.1 Escenarios para disciplina de requisitos

La metodología para desarrollar la solución propone 4 escenarios. Se estará haciendo uso del **escenario 4**, el cual utiliza Historias de Usuario (HU) como técnica de encapsulación de requisitos. Este escenario se aplica por las características del proyecto luego de haber evaluado el negocio a informatizar. El cliente que es la Dirección General de Producción está relacionado con el equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. En cada iteración se repite el flujo de trabajo de las disciplinas, Requisitos, Análisis y diseño, Implementación y Pruebas internas. De esta forma se brinda un resultado más completo para un producto final de manera creciente. Para llegar a lograr esto, cada requisito debe tener un completo desarrollo en una única iteración.

2.2 Requisitos de software

La identificación de los requisitos del software permite interpretar las condiciones o capacidades que debe cumplir el sistema para que se ajuste a las necesidades del cliente. Los requisitos del software son clasificados en dos grupos: Requisitos funcionales y Requisitos no funcionales. La ingeniería de requisitos se define como el uso sistemático de procedimientos, técnicas, lenguajes y herramientas para obtener el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo que satisfaga las necesidades del usuario. (Pressman, 2010)

La ingeniería de requisitos permite una correcta descripción de los requisitos para evitar futuros errores y reducir el tiempo en el desarrollo de un software. De igual manera tributa a mejorar la calidad del software, proporcionando una guía de lo que el usuario

desea, de modo que si se cumplen estos requisitos se satisfacen las necesidades del cliente.

2.2.1 Técnicas para captura de requisitos

Para la captura de requisitos de la solución propuesta se emplearon las técnicas de entrevista y la observación.

- **Entrevistas**: Es un medio tradicional de obtención de requisitos. La entrevista es un método muy efectivo que permite conocer los problemas de los clientes y encontrar requisitos generales. (Ganesh, 2008) Se aplicaron entrevistas a los especialistas del proyecto con los roles de analista, arquitecto y desarrollador, para la documentación posterior del producto de trabajo Historias de Usuarios.
- **Observación**: Este método consiste en la identificación de requisitos cuando se observan a las personas haciendo su trabajo diario. Es muy usado para encontrar requisitos adicionales cuando el usuario es incapaz de explicar los requisitos que necesita para el nuevo sistema. (Ganesh, 2008) Específicamente el tipo de observación realizada fue la observación asistemática. Se aplica este método para realizar los diseños posteriores de las interfaces de usuarios y los comportamientos de las operaciones que se deben de ejecutar.

2.2.2 Requisitos funcionales

Los requisitos funcionales (RF) permiten identificar las condiciones que debe cumplir un sistema para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta. (Pressman, 2010) Una vez obtenidos los requisitos funcionales que debe cumplir la solución, fueron agrupados siguiendo los criterios siguientes:

- Agrupación funcional. Se agrupan los requisitos que tienen una relación funcional directa según el tipo de datos que van a manejar o según los procesos de negocio que van a cubrir. Se aplica agrupando por relación funcional directa, ejemplo excepciones para adicionar, eliminar y modificar. Como resultado se obtiene gestionar excepciones.
- Agrupación modular. Se divide el sistema en módulos y luego se identifica a qué modulo afecta cada requisito. Esta división permite detectar el alcance de cada requisito, delimitando las partes del sistema o componentes a los que afecta.

Componente Caché: **RF1:** Configurar tipo de caché **RF2:** Limpiar caché **Componente Excepciones:** **AGRF1:** Gestionar Excepciones - **RF3:** Adicionar excepciones - **RF4:** Modificar excepciones - **RF5:** Eliminar excepciones - **RF6:** Buscar excepciones - **RF7:** Listar excepciones **Componente Trazas:** **RF8:** Activar o desactivar trazas **RF9:** Listar tipos de trazas **RF10:** Buscar trazas **RF11:** Listar trazas **RF12:** Eliminar trazas por criterio de selección	**Componente Aspectos:** **AGRF2:** Gestionar Aspectos - **RF13:** Adicionar aspectos - **RF14:** Modificar aspectos - **RF15:** Eliminar aspectos - **RF16:** Buscar aspectos - **RF17:** Listar aspectos **Componente IUX:** **RF18:** Configurar parámetros **RF19:** Mostrar vista previa del header

2.2.3 Requisitos no funcionales

Los requisitos no funcionales (RNF) permiten definir las propiedades que el producto debe tener. Estas características son las que hacen el producto más atractivo, usable, rápido y confiable. Una vez que se conozca lo que el componente debe hacer será posible determinar cómo ha de comportarse y qué cualidades debe tener. Dichas propiedades son importantes para los clientes y usuarios a la hora de valorar cuán aceptable es el producto. (Barrios Iglesias, et al., 2013) A continuación se presentan los requisitos no funcionales siguiendo la categoría de clasificación.

Categoría de Usabilidad

RNF1. La capacidad del producto de software de ser amigable para el usuario.

RNF2. Requiere Angular-Material para la representación de los elementos visuales mejorando en el diseño de las interfaces siendo más intuitivas para el usuario.

Categoría de Software

PC Cliente

RNF3. Requiere un navegador web instalado. En el caso de Mozilla Firefox la versión tiene que ser superior a la 40.

PC Servidor

RNF4. Servidor Apache instalado.

RNF5. Gestor de bases de datos instalado.

RNF6. Lenguaje de programación PHP con las extensiones ldap, mcrypt.

2.2.4 Técnicas para validación de requisitos

Existen varias técnicas para validar los requisitos, las cuales se aplican con el objetivo de examinar las especificaciones para asegurar que todos los requisitos han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones y que los errores detectados hayan sido corregidos (Pressman, 2010). Las técnicas a emplear para validar los requisitos de la solución son:

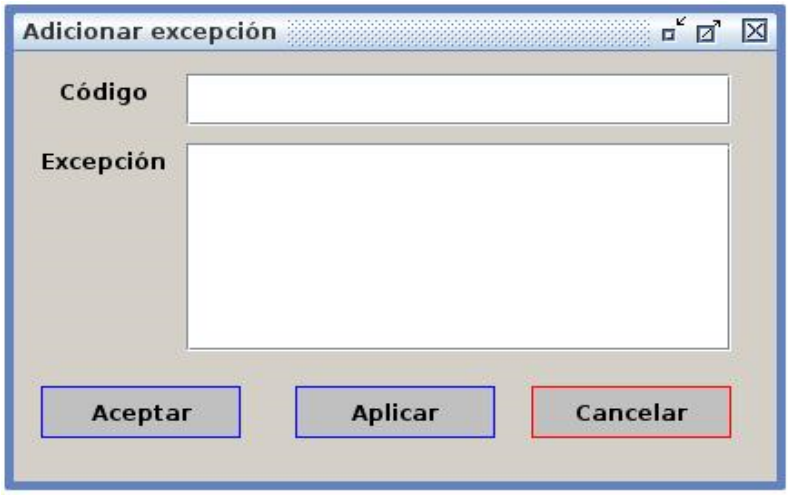
- Revisiones: los requisitos son analizados sistemáticamente por un equipo de revisores, en busca de anomalías y/u omisiones. (Sommerville, 2005) Se aplica dejando constancia en las minutas de reuniones del proyecto y en las revisiones con el cliente y equipo de desarrollo.
- Prototipos: se muestra un modelo ejecutable del sistema a los usuarios finales y los clientes, para que puedan ver si dicho modelo cumple con sus necesidades reales. (Sommerville, 2005) Se aplica mediante la construcción de las interfaces en AngularJS.

El principal resultado de aplicar las técnicas de manera sistemática consiste en la constante actualización y refinamiento de los requisitos, lo cual tributa al desarrollo de una solución eficaz.

2.3 Historias de Usuario

A continuación es presentada la historia de usuario del requisito funcional Adicionar excepción por componente, el cual se encuentra embebido en el requisito Gestionar excepciones. Una parte de las historias de usuario de los requisitos se encuentran en la sección Anexos del 3 hasta el 7, el resto de HU pueden ser encontradas en el expediente de proyecto.

Número: 79	**Nombre del requisito:** Adicionar excepción por componente

Programador: Daniel Herrera Sánchez	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: Poca experiencia de los estudiantes en las tecnologías de desarrollo del proyecto.	**Tiempo Real:** N/A
Descripción: Este requisito se encarga de adicionar una nueva excepción. De una excepción se debe conocer el código y el tipo de excepción. Luego de insertado los campos el sistema efectuará las validaciones pertinentes, en caso que existan errores muestra un mensaje indicando los campos incorrectos. Si toda la información brindada es correcta se muestra un mensaje indicando el éxito de la operación.	
Observaciones: N/A	
Prototipo de interfaz: 	

2.4 Diagrama de clases de diseño

En el diagrama se presentan las clases de la implementación y las relaciones entre las mismas que tiene el componente IUX. El mismo cuenta con un conjunto de páginas web, formularios y librerías que son las encargadas de presentarle al usuario la información referente al componente. Cuando el usuario interactúa con estos elementos el servidor se comunica con el núcleo de la aplicación (AppKernel), el cual envía los datos de la petición del usuario al controlador (ConfigController). Este controlador se vale de los diferentes métodos que tiene definidos para procesar dichos datos y brindarle una respuesta al usuario inicial. De manera similar ocurre con el resto de los componentes.

Los diagramas de diseño de los mismos se encuentran en la sección Anexos desde el 8 hasta el 11.

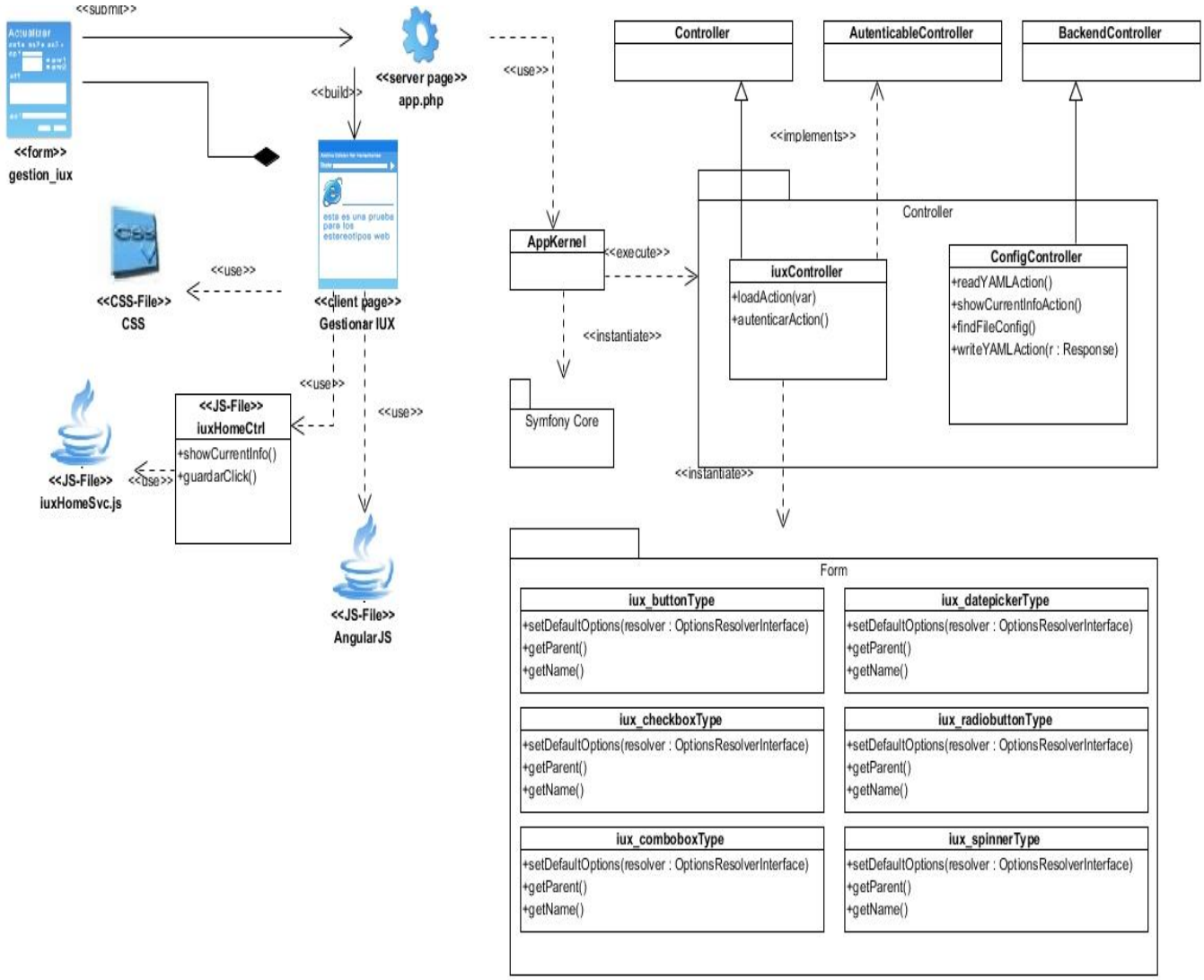


Figura 2. Diagrama de Clases de Diseño del Componente IUX

2.5 Estilos y patrones arquitectónicos utilizados

Patrón Modelo-Vista-Controlador

Uno de los patrones utilizados en la solución propuesta es el Modelo-Vista-Controlador. El marco de trabajo Bosón emplea este patrón para organizar los elementos de sus componentes. AngularJS también establece este patrón para la organización de los elementos que conforman la interfaz gráfica. Específicamente en el caso del componente IUX estos elementos son: página *GestionarIUX.html* como Vista y el *iuxHomeCtrl.js* como Controlador. En este caso el Modelo es el elemento encargado de unir la Vista con el Controlador a través de la etiqueta *model* y objeto *$scope*.

2.6 Patrones de diseño

Patrón Experto en información

Este patrón indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). Para el caso del componente IUX este patrón se emplea en la implementación de las funcionalidades de los controladores tanto de AngularJS como del componente. AngularJS tiene en su controlador los métodos para acceder a la información que está disponible en la vista, mientras que el controlador *ConfigController* es quien tiene las operaciones para modificar el fichero de configuración del componente que se encuentra alojado en el lado del servidor.

Patrón Controlador

El patrón controlador plantea que los controladores sirven como intermediarios entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es quien recibe los datos del usuario y los envía a las distintas clases según el método en cuestión. Este patrón sugiere que la lógica de negocio debe estar separada de la capa de presentación para aumentar la reutilización de código. En el caso del componente IUX este patrón se muestra en los controladores de AngularJS y del componente. El controlador de AngularJS es el encargado de manejar toda la información que se obtiene de la vista, liberando a la misma de cualquier tipo de responsabilidad en el tratamiento los datos. De manera similar ocurre con el *ConfigController*, el cual controla las operaciones de la configuración del componente.

Patrón Alta Cohesión

Este patrón indica que la información que almacena una clase debe de ser coherente y debe estar relacionada con la misma. En el caso del componente IUX esto se observa en todas las clases nuevas que se adicionan. La vista por su parte solo tiene los elementos relacionados con el componente y su único propósito es mostrar y recibir la información. Los controladores por su parte implementan operaciones para darle el tratamiento a estos datos, cada cual en su nivel correspondiente, y solo manejan la información del componente en cuestión.

Patrón Bajo Acoplamiento

Este patrón se propone mantener al mínimo las relaciones de usos entre clases. En caso de producirse una modificación en alguna de ellas, debe provocar poca repercusión en el resto de clases, potenciando la reutilización y disminuyendo la

dependencia entre las mismas. Los nuevos elementos del componente IUX se proponen mantener esta línea al tener solo una relación de uso entre a vista y el controlador de AngularJS, y no tener relación de uso ninguna asociada al *ConfigController*. De esta manera pueden ser sustituidos cualquiera de estos elementos que no se afecta en gran medida el funcionamiento del componente. Una correcta aplicación del patrón de diseño experto en información tributa a un bajo acoplamiento, puesto que ningún método va a intentar modificar información que se encuentre en alguno de los elementos mencionados.

2.7 Modelo de Datos

El modelo de datos es un conjunto de conceptos que sirven para describir la estructura, la semántica y las relaciones que existen entre las entidades de una base de datos, mostrando los datos que serán contenidos en el sistema (Fernández, 2011). El modelo de datos no solo se limita a las bases de datos, también es utilizado para representar los ficheros que se modifican en una operación, en el cual exista persistencia de información. En el caso del componente IUX el fichero de configuración, en el cual tendrá lugar la persistencia de la información puede tener cualquier nombre, y puede estar localizado en cualquier sitio de la aplicación, aunque se debe respetar la estructura presentada en la Figura 3. El resto de los modelos de datos de la solución se encuentran en la sección de Anexos desde el 12 hasta el 15.

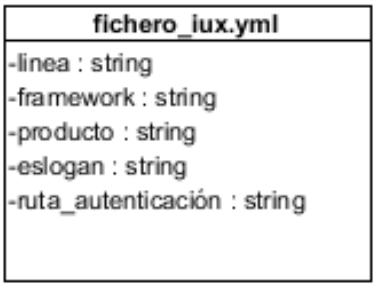


Figura 3. Estructura del fichero de configuración para IUX

2.8 Métricas de validación del diseño

La métrica TOC está dada por la cantidad de operaciones o métodos asignados a una clase y evalúa los atributos **Responsabilidad**, **Complejidad de implementación** y **Reutilización**. Para el caso del componente IUX se analizaron un conjunto de 3 clases, obteniendo el valor 3.3 como promedio en cantidad de procedimientos. Luego de aplicar los criterios de medición correspondientes a la métrica se obtienen los siguientes resultados:

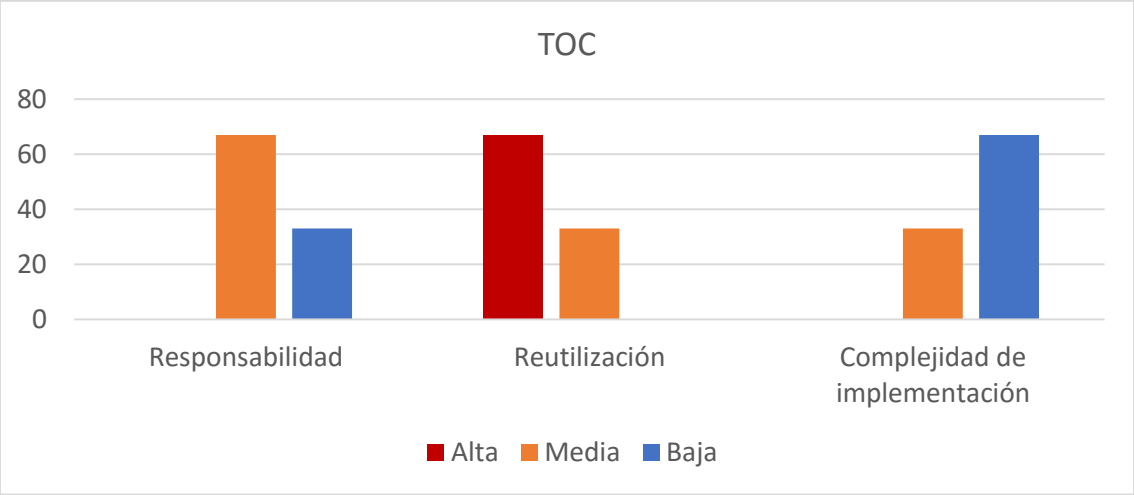


Figura 4. Resultados de aplicar métrica TOC

La métrica RC está dada por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad: **Acoplamiento**, **Complejidad de mantenimiento**, **Reutilización** y **Cantidad de pruebas**. Para el caso del componente IUX se analizaron un conjunto de 3 clases, obteniendo el valor 0.3 como promedio de relaciones de uso entre las mismas. Luego de aplicar los criterios de medición correspondientes a la métrica se obtienen los siguientes resultados:

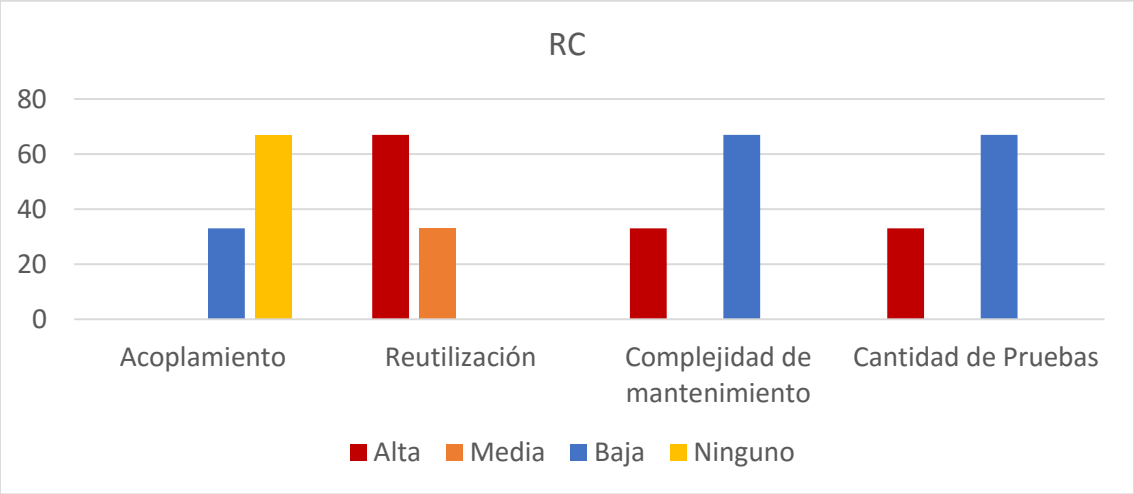


Figura 5. Resultados de aplicar métrica RC

Analizando los resultados obtenidos luego de aplicadas las métricas es posible observar que el diseño ofrece una responsabilidad media, baja complejidad de implementación, alta reutilización, bajo acoplamiento, baja complejidad de mantenimiento y un bajo indicador en cantidad de pruebas.

Los resultados obtenidos indican que aproximadamente el 67% de las clases del componente se encuentran desacopladas, lo cual se valora de positivo para el diseño de la solución pues existen muy pocas relaciones de uso entre los nuevos elementos que se adicionan. Esto trae como consecuencia que al existir cambios en la solución la

repercusión en el resto de los elementos es mínima. El bajo acoplamiento de elementos favorece a la reutilización, la cual es alta en un 67% de las clases del componente. Por otra parte, la complejidad de mantenimiento, de implementación, así como la cantidad de pruebas arrojan valores mayormente bajos. Esto quiere decir que se requiere de poco esfuerzo en estas tareas.

Teniendo en cuenta los resultados obtenidos y lo que representan cada uno para la solución, es posible afirmar que el diseño propuesto tributa al desarrollo de una aplicación con calidad.

2.9 Conclusiones del capítulo

Luego de haberle dado cumplimientos a los objetivos trazados para el segundo capítulo, se arribaron a las siguientes conclusiones:

- Fueron diseñadas las interfaces gráficas de usuarios de los componentes, representando las clases y las relaciones entre las mismas para una mejor definición de la implementación.

- La definición de la arquitectura del sistema mediante el patrón arquitectónico MVC permitió delimitar la vista, en la cual se construye las interfaces gráficas de usuarios, la lógica del negocio para la definición de los controladores que gestionan las operaciones de los componentes, y el modelo para la persistencia de la información.

Capítulo 3: Implementación y prueba

Introducción

En el presente capítulo se describen los estándares de codificación empleados durante la implementación de las interfaces gráficas de los componentes. Se muestran los resultados de las pruebas efectuadas al software y se presenta la estrategia que se sigue para la validación de la investigación.

3.1 Estándares de codificación

Para un proyecto de desarrollo de software se definen estándares de codificación debido a que un estilo de programación homogéneo permite que todos sus participantes puedan entenderlo en menos tiempo. Esto trae como consecuencia que el código sea legible y mantenible. (Eguiluz, 2014)

Los marcos de trabajo Bosón y Symfony establecen el uso de los estándares de codificación PSR-0, PSR-1 y PSR-2. Algunos de los elementos generales contemplados en estos estándares son:

- Los archivos deben utilizar solamente las etiquetas <?php y <?=.
- Los archivos deben emplear solamente la codificación UTF-8 para el código PHP.
- Las constantes de las clases deben declararse en mayúsculas con guiones bajos como separadores.
- El código debe usar cuatro espacios como indentación, no tabuladores.
- La visibilidad debe estar declarada en todas las propiedades y métodos; abstracto y final deben estar declaradas antes de la visibilidad; estático debe estar declarada después de la visibilidad.

3.1.1 Estándar para clases

Después de incluir los **namespace** y los **use**, se añade la información de la clase de la siguiente manera:

```
/**
 * Descripción
 *
 * @author nombre  <usuario@dominio.dom>
 */
```

La información irá contenida entre los signos /** */. Se declaran 2 bloques, separados y diferenciados entre sí por un caracter de espacio *. En el primer bloque se ofrece una

breve descripción del propósito de la clase. En el segundo bloque se ofrecen los datos del autor de la clase. Esta información del autor es opcional.

3.1.1 Estándar para métodos

La estructura que debe añadirse cuando se declara un método o función:

```
/**
 * Descripción
 *
 * @param tipo $variable
 * @param tipo $variable2
 *
 * @return tipo
 */
  private function miFuncion($variable, $variable2){...}
```

La información está contenida entre los signos /** */. Se declaran 3 bloques de datos, cada uno con sus funciones específicas. Los bloques se diferencian y separan entre sí por un caracter de espacio *. En el primer bloque se ofrece una breve descripción del propósito del método o función. En el segundo bloque se especifican los parámetros que recibe, especificando para cada uno su tipo y su nombre. En el tercer bloque se especifica el o los resultados que devuelve el método o función. Los nombres de los métodos deben ser declarados utilizando la notación camelCase.

3.2 Modelo de componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre los mismo. Los componentes físicos incluyen archivos, módulos, ejecutables, paquetes, entre otros. Los diagramas de componentes prevalecen en el campo de la arquitectura de software, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. (Sommerville, 2005) En el caso del componente IUX se observa que el mismo no requiere ningún servicio para funcionar correctamente, y que provee de 4 servicios que pueden ser consumidos por otro componente externo que se integre al marco de trabajo Bosón. El resto de los diagramas de componentes se encuentran en la sección de Anexos del 16 al 19.

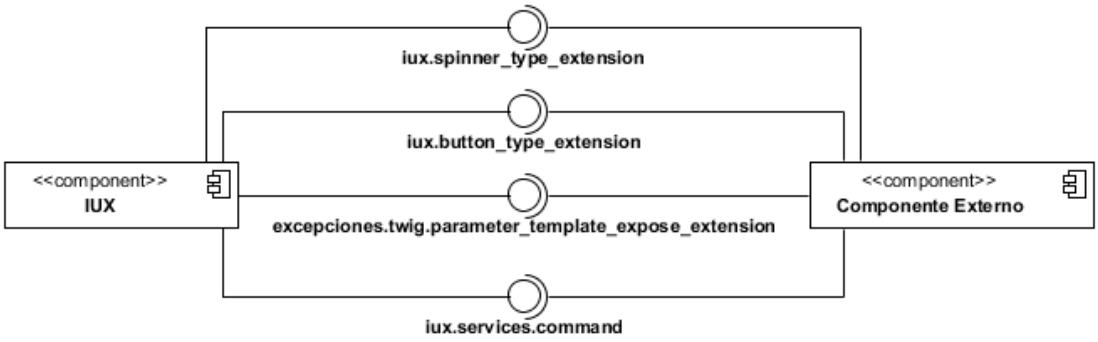


Figura 6. Modelo de componente para IUX

3.3 Pruebas de software

Las pruebas de software son un elemento crítico para la garantía de la calidad de la aplicación y representan una revisión final de las especificaciones, del diseño y de la codificación. Las mismas son realizadas con el objetivo de detectar errores en el sistema, por lo que se llevan a cabo durante todo el ciclo de vida del producto. Estas tienen como objetivo, además de descubrir errores, medir el grado en que el software cumple con los requerimientos definidos. (Pressman, 2010)

Con el objetivo de validar el correcto funcionamiento de las interfaces gráficas de usuarios se aplicaron pruebas de caja negra, utilizando el método de partición equivalente. Además, se aplican pruebas de integración para definir la forma de integración de los componentes con Bosón, funcionando como un todo.

3.3.1 Pruebas de caja negra

Las pruebas de caja negra hacen referencia a pruebas que se llevan a cabo sobre la interfaz del software sin tener en cuenta el código de la aplicación. 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. Según (Pressman, 2010) estas pruebas tienen como propósito detectar:

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

Figura 7. - Interpretación gráfica de las pruebas de caja negra

Las pruebas de caja negra comprenden un conjunto de diferentes técnicas como, por ejemplo: Partición de Equivalencia, Análisis de Valores Límites y Grafos de Causa-Efecto. De estas técnicas se selecciona para ser utilizada la **Partición de Equivalencia**, la cual divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. Para la aplicación de la misma se realizaron 19 diseños de casos de prueba (DCP) uno por cada requisito funcional definido, los cuales se basan en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. (Pressman, 2010)

Como resultado de la aplicación de las pruebas de caja negra fueron identificadas un total de 67 No Conformidades (NC), las cuales se fueron corrigiendo en tres iteraciones. La relación de NC por componente puede ser representada de la siguiente forma:

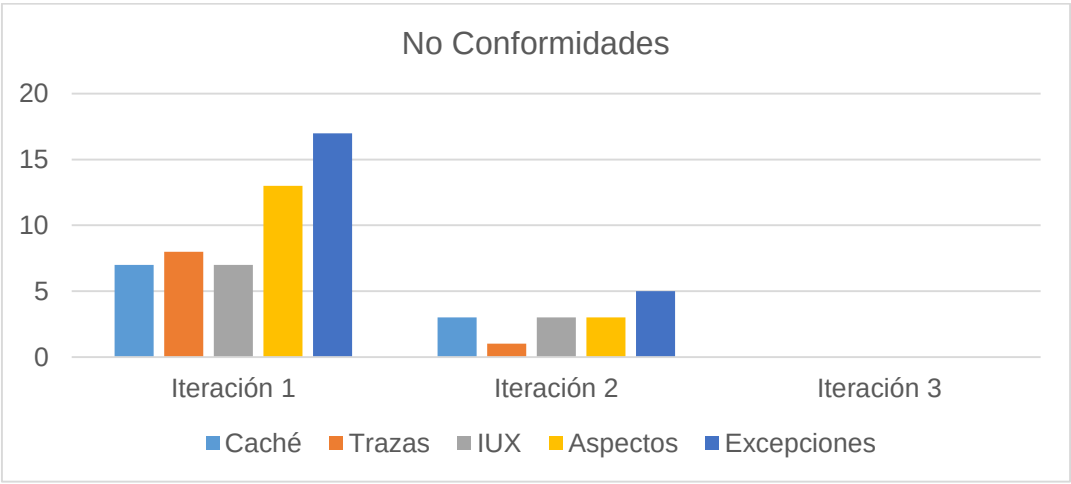


Figura 8. No Conformidades detectadas aplicando pruebas de Caja Negra

3.3.2 Pruebas de integración

La prueba de integración descendente es un planteamiento incremental a la construcción de la estructura de programas. Se integran los componentes moviéndose por la jerarquía de control comenzando por el módulo principal. De acuerdo a (Pressman, 2010) la integración descendente puede ser representada de la siguiente manera:

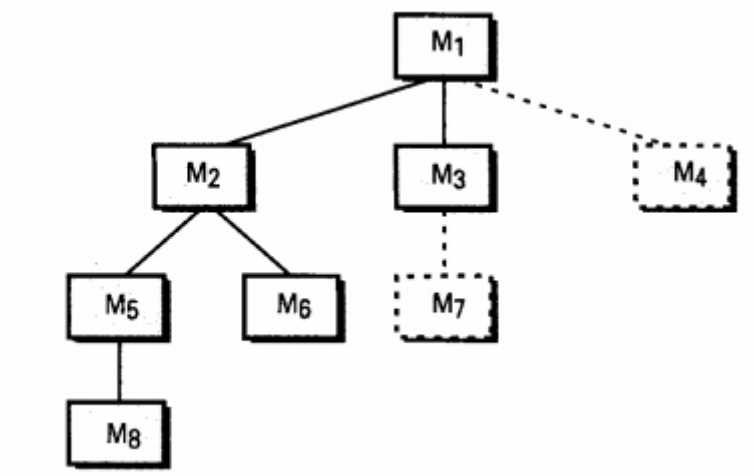


Figura 9. Representación de integración descendente

Los componentes Trazas, Excepciones, Aspectos, IUX y Caché se encontraban implementados e integrados al módulo principal que es el marco de trabajo Bosón, mediante la utilización del núcleo del proyecto que es la clase *AppKernel*. La integración se realiza de forma descendente siguiendo el criterio primero en anchura para cada uno de los componentes. Además, se integran las interfaces gráficas de usuarios para el funcionamiento del sistema como un todo. La siguiente figura presenta cómo quedan integrados los componentes a Bosón.

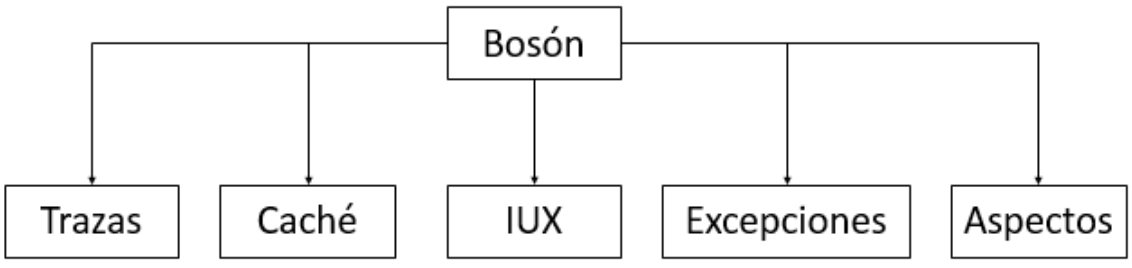


Figura 10. Integración de componentes en Bosón

3.4 Pruebas de aceptación

Estas pruebas permiten que el cliente valide todos los requisitos y las realiza el usuario final en lugar del responsable del desarrollo del sistema. Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie

de pruebas bien planificadas. La prueba de aceptación puede tener lugar a lo largo de semanas o meses, descubriendo así errores acumulados que pueden ir degradando el sistema. (Pressman, 2010) La aplicación de las pruebas de aceptación arrojó como resultado el producto de trabajo: Acta de aceptación del producto, donde se emite una evaluación de la revisión de los productos de trabajos propuestos en la metodología y la entrega del código fuente de la aplicación libre de errores.

3.5 Validación de la investigación

Como parte de la estrategia de validación de la solución propuesta se ha realizado un diseño pre-experimental midiendo la variable dependiente usabilidad. Para la realización de un pre-experimento se definen diseños para pre-prueba y post-prueba. El diseño de pre-prueba permite obtener una referencia inicial y conocer el nivel de la variable dependiente antes de ser estimulada. El diseño post-prueba permite establecer una comparación con los datos obtenidos anteriormente, y el comportamiento de la variable dependiente luego de recibir el estímulo. (Guido Elmer, 2014)

Para la aplicación del diseño experimental se utilizó la Lista de Chequeo Usabilidad de Sitios Web propuesta en el departamento de calidad de la Universidad de las Ciencias Informáticas como instrumento de medición, la cual está compuesta por un conjunto de 144 preguntas enfocadas a medir indicadores de usabilidad (ver Anexo 20). El resultado de aplicar la pre-prueba muestra el siguiente comportamiento para la variable usabilidad:

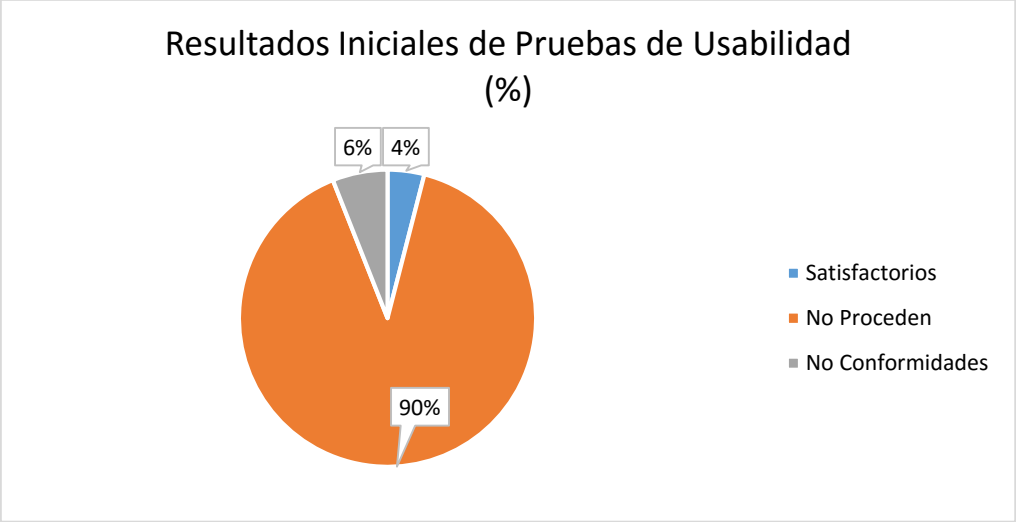


Figura 11. Resultados iniciales de pruebas de usabilidad

En la gráfica se puede observar que hay un porciento muy elevado de preguntas evaluadas como No Proceden (90%). Esto se debe a que las listas de chequeo de usabilidad están enfocadas en interfaces gráficas de usuario, las cuales para la configuración de componentes del marco de trabajo Bosón eran inexistentes. Luego se encuentran las No Conformidades con un 6% sobre un 4% de elementos evaluados de

manera satisfactoria, demostrando que existe un bajo índice de usabilidad en los componentes evaluando los indicadores propuestos.

El resultado de aplicar la post-prueba muestra el siguiente comportamiento para la variable usabilidad:

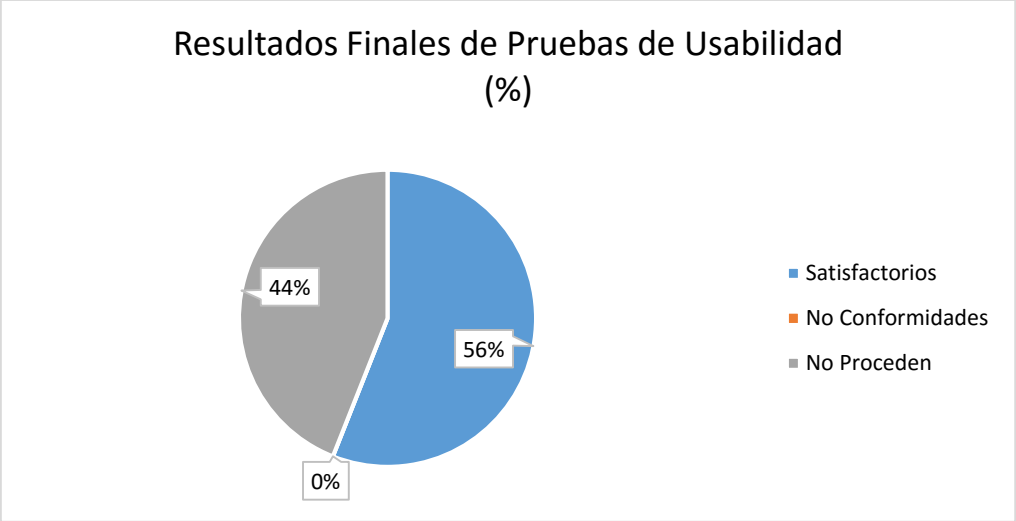


Figura 12. Resultados finales de pruebas de usabilidad

Esta revisión estuvo dividida en dos iteraciones. En una primera iteración fueron detectadas tres No Conformidades, las cuales fueron corregidas en una segunda iteración. Como resultado final se obtuvieron 64 de las preguntas calificadas como No Proceden, dejando un total de 80 aspectos evaluados de manera satisfactoria y cero No Conformidades. Esto demuestra que la solución cumple con los indicadores de usabilidad propuestos. Además, supone una mejora en cuanto a usabilidad sobre el estado inicial de los componentes del marco de trabajo Bosón. Lo mencionado con anterioridad se evidencia en el Acta de Liberación de la aplicación (ver Anexo 21).

3.6 Conclusiones del capítulo

Luego de haberle dado cumplimientos a los objetivos trazados para el tercer capítulo, se arribaron a las siguientes conclusiones:

- La implementación de la interfaz gráfica de usuario permitió visualizar y gestionar los parámetros de configuración de los componentes, construyendo así interfaces intuitivas para el usuario.
- La ejecución de las pruebas de caja negra permitió demostrar que las funciones del software son operativas, a través de los diseños de casos de pruebas.
- La prueba de integración descendente permitió demostrar que los componentes funcionan de forma independiente e integrados al marco de trabajo Bosón.

- Fueron aplicadas listas de chequeo de usabilidad sobre la aplicación, con las cuales se obtuvo como resultado que la solución propuesta cumple con estándares requeridos y se mejoran los indicadores de usabilidad para los componentes en cuestión.

Conclusiones

El desarrollo de la investigación permitió arribar a las siguientes conclusiones:

- La fundamentación de las tecnologías permitió indicar que la mayoría de los marcos de trabajos que se utilizan para el desarrollo de las aplicaciones web no implementan interfaces gráficas de usuarios. Los mismos configuran la información de los componentes mediante clases y métodos, lo que no representa una alternativa de solución al problema de la investigación.
- La fundamentación y utilización de la metodología permitió guiar el proceso de desarrollo de las interfaces gráficas de usuarios de los componentes Caché, Excepciones, IUX, Aspectos y Trazas en el marco de trabajo Bosón.
- La fundamentación de las herramientas permitió identificar las potencialidades que brindan para la construcción de las interfaces gráficas de usuarios de los componentes Caché, Excepciones, IUX, Aspectos y Trazas en el marco de trabajo Bosón.
- El análisis y diseño de la propuesta de desarrollo de las interfaces gráficas de usuario de los componentes Caché, Excepciones, IUX, Aspectos y Trazas en el marco de trabajo Bosón permitió construir los productos de trabajos que dieran paso a la ejecución correcta de la implementación.
- La implementación de la propuesta de desarrollo de las interfaces gráficas de usuarios de los componentes Caché, Excepciones, IUX, Aspectos y Trazas en el marco de trabajo Bosón permitió mejorar la usabilidad de los componentes en cuanto a los indicadores evaluados. Queda demostrado en la validación de la investigación, utilizando como instrumento de medición la Lista de Chequeo Usabilidad de Sitios Web.
- Las pruebas de caja negra utilizando la técnica de partición equivalente y las pruebas de integración, permitieron demostrar que el sistema se encuentra libre de errores y se puede integrar como un todo en el marco de trabajo Bosón.

Bibliografía

Acevedo Osorio, Liesner and Osorio Ramirez, Karel. 2011. *Técnicas de compilacón: Manual Practico para estudiantes de informatica.* La Habana : s.n., 2011.

Alexander, Christopher, Ishikawa, Sara and Silverstein, Murray. 1979. *A Pattern Language Towns, Buildings , Construction.* Houston : s.n., 1979.

AngularJS. 2015. [En línea] 2015. http://docs.angularjs.org.

Barchini, Graciela Elisa. 2005. *Método I+D de la Informática.* Argentina : s.n., 2005. 1667-8338.

Barrios Iglesias, Claudia and Pérez Cintrón, Fernando. 2013. *Sistema de análisis web para el entorno virtual del Sistema de Apoyo a la Municipalización en el Ministerio del Poder Popular para la Educación Universitaria en Venezuela.* La Habana : s.n., 2013.

Bieman, J M and Ott, L M. 1994. *Measuring Functional Cohesion.* 1994.

Briggs, Owen. 2003. *Cascading Style Sheets.* 2003.

Calás Torres, Abraham. 2014. *Documentación de Bosón.* [En línea] 2014. http://docs.prod.uci.cu/online/Boson.docset/Contents/Resources/Documents/docs/index.html.

—. 2012. *Definición de pautas para la creación de un modelo de componentes para el marco de trabajo Sauxe.* La Habana : Serie Científica, 2012. Vol. V.

—. 2014. *Modelo de componentes para el marco de trabajo Sauxe.* 2014.

Card, D and Glass, N R. 1990. *Measuring Software Design Quality.* s.l. : Prentice-Hall, 1990.

Chapman, Stephen. 2007. *JavaScript.* [En línea] 2007. http://javascript.about.com/od/reference/a/history.htm.

Crnkovic, Ivica and Larsson, Magnus. 1969. *Building Reliable Component-Based Software Systems.* s.l. : Artech House, 1969. ISBN: 1-58053-327-2.

Day, Robert A. 2005. *Cómo escribir y publicar trabajos científicos.* 3rd. Washington, DC : The Oryx Press, 2005. ISBN 92-75-31598-1.

DesarrolloWeb. 2015. Métricas de Software. [Online] 2015. http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html.

Dhama, H. 1995. *Quantitative Models of Cohesion and Coupling in Software.* s.l. : Journal of Systems and Software, 1995.

Dojo Toolkit. 2015. *Dojo Toolkit.* [En línea] 2015. http://dojotoolkit.org.

Eguiluz, Javier. 2015. *Symfony 2.* [En línea] 2015. http://symfony.es.

—. 2013. *Desarrollo Web Ágil con Symfony 2.* Madrid : s.n., 2013.

—. 2014. Manual de Symfony 2: Estándares de codificación. 2014.

Exton, Chris, Watkins, Damien and Thompson, Dean. 2009. *Comparison between CORBA IDL & COM/DCOM MIDL.* Australia : s.n., 2009.

Ferreira de Lucena Jr, Vicente. 2002. *Flexible Web-based Management of Components for Industrial Automation.* 2002.

Grady, R B and Caswell, D L. 1987. *Sotware Metrics: Establishing a Company-Wide Program.* s.l. : Prentice-Hall, 1987.

Guido Elmer, Jacinto. 2014. *Diseños experimentales de investigación: pre-experimentos y cuasi-experimentos.* 2014.

Hassan Montero, Yusef. 2004. *Propuesta de adaptación de la metología de diseño centrado en el usuario para el desarrollo de sitios web accesibles.* Madrid : s.n., 2004.

Heineman, G. T. and Councill, W. T. 2001. *Definition of a software component and its elements.* Boston : Addison-Wesley, 2001.

Hernández León, Rolando Alfredo and Coello Gonz, Sayda. 2006. *El proceso de investigación científica.* La Habana : Editorial Universitaria, 2006.

Instituto Nacional de Estadística e Informática. 1999. *Herramientas CASE.* 1999.

JetBrains SRO. 2015. *JetBrains.* [En línea] 2015. http://www.jetbrains.com.

La situación problémica como componente del diseño. **García, María Elena Guardo y Carreño Vega , José Enrique. 2010.** 4, Matanzas : s.n., 2010, Revista Pedagogía Universitaria, Vol. XV. ISSN 1609-4808.

Lapuente, Lamarca and Jesús, María. 2008. *Hipertexto: El nuevo concepto de documento en la cultura de la imagen.* [Online] 2008. http://www.hipertexto.info/documentos/html.htm..

Lorenz and Kidd. 1994. *Metrics for Object-Oriented Software Engineering.* 1994.

Los Componentes del diseño teórico de la investigación científica. Una reflexión Praxiológica. **García., María Elena Guardo. 2009.** 3, Matanzas : s.n., 2009, Revista Pedagogía Universitaria, Vol. XIV, pág. 9.

Macleod, M. 1994. *Usability: Practical Methods for testing and Improvemen.* Noruega : s.n., 1994.

McCabe, T J and Watson, A H. 1994. *Software Complexity.* 1994.

McCall, J, Richards, P and Walters, G. 1977. *Factors in Sotware Quality.* 1977.

Object Management Group. 2006. *CORBA Component Model Specification.* 2006.

—. 2003. *MDA Guide Version 1.0.1.* 2003.

Phalcon Technologies. 2015. Reading Configurations in Phalcon. [En línea] 2015. https://docs.phalconphp.com/en/latest/reference/config.html.

PhalconPHP. 2016. PhalconPHP. *PhalconPHP.* [Online] 2016. https://phalconphp.com/es/.

Pressman, Roger S. 2010. *Software Engineering, a Practitioner's Approach.* 7th Edition. s.l. : McGraw-Hill, 2010. 978-0-07-337 597 -7.

Pupo Leyva, Iliannis. 2012. *Procedimiento para el aseguramiento y evaluación de la usabilidad basado en patrones arquitectónicamente sensibles para los sistemas de gestión del Centro de Informatización de la Gestión de Entidades.* 2012.

Ricardo Pino Torrens. 2006. *Marco teórico referencial de la investigación. .* La Habana : s.n., 2006.

Riehle, Dirk. 2000. *Framework Design: A Role Modelin Approach.* 2000.

Rodríguez Sánchez, Tamara. 2014. *Metodología de desarrollo para la actividad productiva de la UCI.* Habana : s.n., 2014.

Rojas Morales, Carmenza, Alcocer Olaciregui, Adalgisa and Jabba Molinares, Daladier. 2004. *Revista Científica Ingeniería y Desarrollo.* 2004.

Royo, Javier. 2004. *Diseño Digital.* Madrid : s.n., 2004.

Rumbaugh, James, Jacobson, Ivar and Booch, Grady. 1999. *El lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley, 1999.

Sametinger, J. 1997. *Software engineering with reusable components.* s.l. : Springer Verlag, 1997.

Sampieri, Roberto Hernández, Fernández-Collado, Carlos y Lucio, Pilar Baptista. 2006. *Metodología de la investigación.* Mexico DF : McGraw-Hill, 2006. ISBN 970-10-5753-8.

Sauro, Jeff and Kindlund, Erika. 2005. *A Method to Standardize Usability Metrics Into a Single Score.* 2005.

Sencha Inc. 2015. *Sencha.* [En línea] 2015. https://www.sencha.com.

Shackel, B. 1991. *Usability : Context, framework, design and evaluation.* Inglaterra : s.n., 1991.

Shneiderman, Ben. 1998. *Designing the user interface, Strategies for effective Human-computer interaction.* s.l. : Addison-Wesley, 1998.

Sitio web oficial de Apache. 2015. *The Apache Software Foundation.* [En línea] 2015. http://www.apache.org/.

Sitio web oficial del producto Visual Paradigm. 2014. *Sitio web oficial del producto Visual Paradigm.* [En línea] 2014. http://www.visual-paradigm.com/product/vpuml/..

Sitio web oficial MySQL. 2015. *MySQL.* [En línea] 2015. http://www.mysql.com/.

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

Stallings, William. 1998. *The Origins of OSI.* [En línea] 1998. http://williamstallings.com/Extras/OSI.html.

Szyperski, C. 1998. *Component Software. Beyond Object-Oriented Programming.* Boston : Addison-Wesley, 1998.

Trejos Burtica, Omar Ivan. 1999. *La esencia de la lógica de programación.* s.l. : Papiro, 1999. ISBN 958-33-1125-1.

Tutorialspoint. 2014. *Learn AngularJS, web application framework.* [En línea] 2014. http://www.tutorialspoint.com.

Valbuena Aponte, Angela María . 2014. *Guía comparativa de frameworks para los lenguajes HTML5, CSS y JavaScript para el desarrollo de aplicaciones Web.* Pereira : s.n., 2014.

Vélez, Carlos Mario. 2005. *Apuntes de metodología de la investigación.* Medellin : Universidad EAFIT, 2005.

Yii Framework. 2016. Yii Framework. *Yii Framework.* [Online] Yii, 2016. http://www.yiiframework.com/.

Zend Technologies Ltd. 2015. Introduction to Zend\Config. [Online] 2015. http://framework.zend.com/manual/current/en/modules/zend.config.introduction.html.