UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Título: Módulo de gestión de trazas para el marco de trabajo Symfony 2.

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

Autor:

René Olazabal Arteaga.

Tutores:

Ing. Arnolis Salgueiro Arzuaga.

Ing. René Rodrigo Bauta Camejo.

Junio 2014

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor del trabajo titulado:

Módulo de gestión de trazas para el marco de trabajo Symfony2.

Y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente en el mes de _____ del año _____.

René Olazabal Arteaga

_____

Firma del Autor

Ing. Arnolis Salgueiro Arzuaga

_____

Firma del Tutor

Ing. René Rodrigo Bauta Camejo

_____

Firma del Tutor

DEDICATORIA

Dedico este trabajo a mis padres por ser mi guía durante toda mi vida. Por su comprensión, respeto, apoyo y todo su amor, los admiro y estoy orgulloso de tener tan maravillosos padres.

AGRADECIMIENTOS

A mis padres por darme todo su apoyo, cariño y amor. A mi hermana Nailín por todo el sacrificio y esfuerzo que ha hecho durante estos años para que yo pueda salir adelante y encaminarme.

A mi tío Martín por toda su ayuda y sus consejos, a Maribel quien me acogió en su casa durante estos cinco años como si fuera un hijo a pesar de no serlo.

A mis compañeros de grupo que me han acompañado durante estos años, he aprendido mucho con cada uno de ellos. Especialmente a mi equipo de asalto combate como lo nombramos hace ya un tiempo, José Miguel, Irene, Willian, Lizandra, Jaca.

A mis compañeros de equipo los cabezones, Randy, Andrés y Saidel con los que tanto improvisé en seminarios, clases prácticas, etc. Nunca voy a olvidar aquellos ppt de última hora los cuales nos repartíamos durante las exposiciones y todos los momentos que disfrutamos, nos reímos, bailamos, todos los consejos y también los malos porque no, de cada uno aprendí mucho.

A toda la gente del edificio con la que he vivido Alejandro, Yoelvis, Ángel Félix, Ariam, Raciel, Ledian, Alejandro, Pedrito, Osmail, Saname, Yasel y otros que ya no están presentes.

A todos los profesores que me dieron clases y contribuyeron a mi formación, también a los profesores del proyecto Yunet, Yarenis y mis tutores Arnolis y René por su ayuda durante este tiempo.

A mis amigos del edificio 10 Medinilla, Pablo, Andrew, los dos Ernesto, Yaniel, Roberto, Rubén, por siempre estar ahí en los momentos de estrés cuando estaba al borde de la locura y decirme vamos un jueguito para despejar.

RESUMEN

La UCI es una universidad productiva conformada por varios centros especializados en un área determinada de la informática. Especialmente el centro CEIGE, encargado de desarrollar soluciones que apoyen la gestión empresarial, prescinde actualmente de un componente que permita la gestión de las trazas de los sistemas que se implementan sobre el framework de desarrollo Symfony 2. En el afán de intentar mitigar este problema la presente investigación propone un Módulo para la gestión de trazas, específicamente para integrarse a Symfony 2 y con el objetivo de disponer de un componente encargado de realizar tan importante proceso. Este Módulo permite la gestión de 4 tipos de trazas: acción, datos, rendimiento y excepción. Asimismo mediante una funcionalidad que permite conocer todas las trazas capturadas, se ofrece la posibilidad de realizar auditorías al sistema donde se utilice. En un último momento se exponen los resultados obtenidos luego de aplicar pruebas de software al Módulo, permitiendo considerar como satisfactoria la propuesta presentada.

Palabras claves: gestión de trazas, Symfony 2, auditorías, trazas.

ÍNDICE

INTRODUCCIÓN 7
1. CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN. 11
1.1. Conceptos asociados al dominio del problema 11
1.2. Estudio de sistemas informáticos para la captura y gestión de trazas. 12
1.3. Proceso de desarrollo de software 15
1.3.1. Modelo de desarrollo de software 15
1.4. Tecnologías utilizadas en el desarrollo de la propuesta de solución 16
1.4.1. Framework de desarrollo 16
1.4.2. ORM 17
1.4.3. Lenguajes de programación. 18
1.5. Herramientas utilizadas para el desarrollo de la propuesta de solución 19
1.5.1. Herramientas Case 19
1.5.2. Sistema Gestor de Base de Datos 20
1.5.3. Entorno de Desarrollo Integrado. 21
1.5.4. Control de versiones 22
1.6. Requisitos de software 23
1.7. Técnicas de captura de requisitos 24
1.8. Técnicas de validación de requisitos 24
1.9. Patrones de diseño 24
1.9.1. Patrones Grasp 25
1.9.2. Patrones Gof 26
1.10. Arquitectura de software 27
1.11. Conclusiones parciales 27
2. CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA 29
2.1. Descripción de la solución propuesta 29
2.2. Modelado del negocio 30
2.3. Modelo conceptual 31
2.4. Requisitos de software 32
2.4.1. Requisitos funcionales 32
2.4.2. Especificación de requisitos funcionales 33
2.4.3. Requisitos no funcionales 37

2.5. Modelado del diseño 37
2.5.1. Patrones de diseño 38
2.5.2. Arquitectura orientada a componentes 39
2.5.3. Patrón de arquitectura 40
2.6. Diagramas de clases del diseño 42
2.7. Diagramas de secuencia 43
2.8. Diagrama de Componentes 44
2.9. Diagrama de despliegue 45
2.10. Conclusiones parciales 45
3. CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN 47
3.1. Métricas de software orientadas a clases para evaluar el diseño 47
3.1.1. Tamaño operacional de clases (TOC) 48
3.1.2. Resultados obtenidos de la aplicación de la métrica TOC 49
3.1.3. Relaciones entre clases 52
3.1.4. Resultados obtenidos de la aplicación de la métrica RC 53
3.2. Pruebas de software 57
3.2.1. Pruebas de caja blanca 58
3.2.2. Resultados de las pruebas de caja blanca 60
3.2.3. Pruebas de caja negra 60
3.2.4. Resultados de las pruebas de caja negra 61
3.2.5. Prueba de aceptación 61
3.2.6. Resultados de las pruebas de aceptación 62
3.3. Conclusiones parciales 62
CONCLUSIONES GENERALES 64
REFERENCIAS BIBLIOGRÁFICAS 66

INTRODUCCIÓN

Con el transcurso del tiempo ha venido desatándose un vertiginoso desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) e Internet, el cual ha traído aparejado grandes avances para la Informática. En la actualidad esta ciencia se ha vinculado a diversas esferas como la educación, la salud, la aeronáutica, la seguridad entre otras, facilitando principalmente los procesos de toma de decisiones y gestión empresarial. Este último consiste en un ciclo permanente para solucionar problemas en una entidad determinada. Su principal objetivo es aumentar la productividad y la competitividad de las empresas, de ahí la importancia de desarrollar sistemas informáticos de alta calidad.

La Universidad de las Ciencias Informáticas (UCI) creada en el año 2002, constituye un centro de altos estudios donde se vinculan el estudio, la investigación y la producción. Formar profesionales comprometidos con su Patria y altamente calificados en la rama de la Informática. Producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación. Servir de soporte a la industria cubana de la informática (Uci, 2014). Para lograrlo es necesario el uso de tecnología y técnicas novedosas, siempre partiendo del paradigma de independencia tecnológica vigente en el país usando software libre. Gracias a esto se han obtenido resultados relevantes en las esferas de educación, tele-formación, salud, bioinformática, automatización, gestión empresarial y otras.

La UCI está compuesta por varios centros productivos, cada uno de ellos se especializa en el desarrollo de sistemas para las diversas esferas sociales. El Centro de Informatización y Gestión de Entidades (CEIGE), cuenta con un Departamento encargado de gestionar las tecnologías utilizadas para el desarrollo de software. Actualmente en dicho Departamento se desea implementar un sistema sobre el marco de trabajo Symfony 2, que permita la creación de software de gestión. En estos sistemas se hace necesario tener conocimiento de los eventos, acciones o excepciones que a diario ocurren, esos procesos se traducen a código fuente y luego se reproducen constantemente, la prueba de que estos procesos han ocurrido son las trazas.

Las trazas son un mecanismo de registro de eventos y datos, o información sobre quién, qué, cuándo y dónde un evento ocurre, ya sea para un módulo de un sistema en particular o para toda la aplicación. Son utilizadas para dejar un registro de todas las funciones que se realizan en un sistema, así como acreditar las validaciones de datos previstas, sin modificar el sistema en ningún momento. También son usadas con el fin de monitorear las acciones que se realicen (acceso a

ficheros, dispositivos, empleo de los servicios, etc.), y para detectar indicios de hechos relevantes a los efectos de la seguridad que puedan afectar la estabilidad o el funcionamiento del sistema informático. Además rastrean los caminos que siguen los datos a través del programa. Contribuyen al fortalecimiento de la seguridad en las aplicaciones web, el registro de las mismas posibilita que dichas aplicaciones sean auditables, fortaleciendo así la seguridad en las aplicaciones.

En el marco de trabajo Symfony 2 existen mecanismos para almacenar determinadas trazas; proceso muy puntual en cada uno de los módulos que resulta muchas veces engorroso para los desarrolladores y que tiene un resultado poco organizado dificultando así su consulta y entendimiento, por lo que se hace necesario el desarrollo de un componente que pueda integrarse a Symfony 2 y permita almacenar las trazas de los sistemas de manera organizada y de fácil acceso.

Lo anterior permite plantear el siguiente **problema a resolver**: ¿Cómo garantizar los mecanismos para el registro organizado y abstracto de las trazas en el marco de trabajo Symfony 2?

Enmarcado en el **objeto de estudio**: Sistemas o módulos de captura y gestión de trazas, delimitándose como **campo de acción**: Captura y gestión de trazas en Symfony 2.

Para dar solución al problema planteado se define el siguiente **objetivo general:** Desarrollar un módulo de gestión de trazas que garantice el registro organizado y abstracto de las mismas en el marco de trabajo Symfony 2.

Desglosándose en los siguientes **objetivos específicos:**

1. Construir el marco teórico-referencial de la investigación a partir del estudio de las particularidades de los sistemas de gestión de trazas.
2. Realizar la modelación del negocio con el fin de sentar las bases para un adecuado levantamiento de requisitos.
3. Implementar las funcionalidades necesarias siguiendo buenas prácticas del desarrollo de software para dotar al marco de trabajo Symfony 2 de un módulo que gestione las trazas.
4. Validar el trabajo realizado a partir de pruebas de software.

Con el propósito de dar cumplimiento a los objetivos definidos se establecen las siguientes **tareas de investigación**:

- Elaboración del modelo conceptual de la investigación.
- Identificación de los requisitos funcionales.

- Descripción de los requisitos identificados.
- Validación de los requisitos identificados.
- Realización del diagrama de componentes del sistema.
- Realización de los diagramas de clases del diseño.
- Definición del modelo de datos.
- Descripción de las clases definidas en el diseño.
- Implementación de las interfaces de usuario.
- Implementación de las funcionalidades del módulo de gestión de trazas.
- Validación de los resultados obtenidos.

Luego de realizadas las tareas de investigación se espera obtener como **posible resultado** un módulo para la gestión de las trazas en el marco de trabajo Symfony 2.

Idea a defender:

Si se desarrolla el módulo de gestión de trazas, se podrá garantizar el registro organizado y abstracto de las mismas dentro del marco de trabajo Symfony 2.

La investigación se guiará a través de los siguientes métodos científicos:

Métodos teóricos:

- Analítico – sintético: a partir de un profundo análisis acerca de las teorías, documentación y herramientas existentes sobre el registro y control de las trazas en las aplicaciones de gestión, se obtendrán los elementos más importantes que hacen referencia a las herramientas realizadas en el lenguaje PHP.
- Inductivo – Deductivo: con la realización de un profundo análisis del trabajo general de las herramientas de registro de trazas, se podrá llegar a conclusiones de su desarrollo e integración al Marco de Trabajo.
- Histórico-Lógico: Permite conocer y comprender el estado del arte de los sistemas informáticos existentes en el mundo que poseen funcionalidades similares a la solución propuesta, tendencias actuales y proyecciones futuras, conociendo así su evolución y desarrollo.

Métodos empíricos:

- Entrevista: Se emplea principalmente en la captura de requisitos. Esto constituye uno de los mayores afluentes del conocimiento para la descripción de los requisitos funcionales del componente de Trazas.
- Observación: El empleo de este método permitió hacer un análisis detallado de las principales características y funcionalidades del módulo de trazas del marco de trabajo Sauxe.

Para una mejor comprensión del presente documento se estructura en tres capítulos fundamentales. A continuación se realiza una breve descripción de cada uno de ellos.

Capítulo 1- Fundamentación teórica: Se realiza un análisis de los principales conceptos asociados a la investigación estudiando algunos de los sistemas que incorporen módulos para la captura y gestión de las trazas, sus antecedentes y tendencias actuales. Mediante un análisis bibliográfico se determinan las tecnologías, herramientas y lenguajes de modelado y programación a utilizar para implementar la solución propuesta.

Capítulo 2- Análisis y diseño del sistema: Se identifican los requerimientos, tanto funcionales como no funcionales del sistema. Se realiza la descripción de cada uno de los requerimientos identificados. Se confecciona el modelo conceptual y los prototipos de interfaz de usuario como técnica para la validación de los requisitos funcionales. También se realiza el diseño del sistema detallando los patrones de diseño utilizados. Además se define la estructura del módulo y se procede a la validación del diseño. Se confecciona el modelo de datos y se define la arquitectura del sistema.

Capítulo 3: Implementación y prueba: Se lleva a cabo la implementación de la solución y se efectúan las pruebas para la validación, en este caso pruebas de caja blanca y de aceptación.

1. CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN.

En el presente capítulo se realizará un estudio del estado del arte relacionado con el problema al que se dará solución. Se verán conceptos, metodologías, tecnologías y software que se utilizan actualmente en el mundo, relacionados con el control y registro de trazas y que podrán ser utilizados en la solución del problema. Se exponen además las principales herramientas, técnicas y una arquitectura a utilizar durante el desarrollo de la propuesta de solución.

1.1. Conceptos asociados al dominio del problema

Con el propósito de mejorar la comprensión del contenido de este capítulo se presentan a continuación las definiciones de los principales conceptos que guiarán la investigación.

Traza

Una traza es la marca dejada por un elemento al desplazarse a una nueva posición. Llevándolo al contexto informático se puede decir que las trazas son una colección de eventos y datos devueltos por una aplicación, que representan el historial o rastro dejado por cierto proceso cuando se ejecuta en un sistema (Corrales Martínez, 2009).

A raíz del concepto anterior se puede entender que un **sistema de gestión de trazas** es aquel que controla los eventos que ocurren dentro de un software determinado, lo cual resulta muy importante para su correcto funcionamiento. Un sistema informático que no posea un componente con esta funcionalidad está expenso a grandes riesgos, entre ellos: la imposibilidad de recuperarse tras una falla del sistema quedar impune un malhechor en caso de ser quebrantada la seguridad del sistema. Sin las trazas puede ser muy difícil, o en ocasiones imposible, averiguar el motivo del fallo de la aplicación (Sacerio Martínez, 2010).

En los ficheros de trazas se almacena toda la actividad y eventos que ocurren en un sistema: ¿quién entra?, ¿qué comandos ejecuta?, ¿qué errores muestran las aplicaciones? etc. A continuación se definen los tipos de trazas implicados en la investigación.

Traza de acción: registra los atributos generales relacionados con la estructura del sistema donde se ejecutó la acción y a qué actividad y proceso de negocio responde. Incluye, además, información sobre quién la ejecutó, en que estructura y desde qué dirección IP (Baryolo Gómez, 2012).

Traza de excepción: se producen como consecuencia de una violación de reglas impuestas por el negocio o por las restricciones tecnológicas del entorno (Baryolo Gómez, 2012).

Traza de datos: está encargada de registrar los atributos relacionados con el nivel de base de datos. Se ejecuta si una acción realiza alguna operación sobre la base de datos (Baryolo Gómez, 2012).

Traza de rendimiento: registra información relacionada con el consumo de memoria, tiempo de ejecución y demás parámetros relacionados con las acciones ejecutadas (Baryolo Gómez, 2012).

Una vez presentado los conceptos más importantes para la investigación se procede a realizar una exploración documental con el objetivo de informar el conocimiento ya producido respecto al control y registro de eventos.

1.2. Estudio de sistemas informáticos para la captura y gestión de trazas.

- **ADO.NET**

ADO.NET es un conjunto de componentes de software que pueden ser usados por los programadores para acceder a datos y a servicios de datos. Es una parte de la biblioteca de clases base que están incluidas en el Microsoft .NET Framework. Es comúnmente usado por los programadores para acceder para modificar los datos almacenados en un Sistema Gestor de Bases de Datos Relacionales, aunque también puede ser usado para acceder a datos en fuentes no relacionales (Microsoft Developer Network, 2014).

ADO.NET 2.0 es la última versión y ofrece nueva funcionalidad integrada de seguimiento de datos compatible con los proveedores de datos de .NET para SQL Server, Oracle, OLE DB y ODBC, así como DataSet de ADO.NET y los protocolos de red de SQL Server (Microsoft Developer Network, 2014).

Para garantizar la compatibilidad entre diferentes tecnologías de gestión de traza, éste es ampliable, de manera que un programador puede realizar un seguimiento de un problema en cualquier nivel de la pila de la aplicación. A pesar que el seguimiento no es una característica exclusiva de ADO.NET, los proveedores de Microsoft aprovechan las API generalizadas de seguimiento e instrumental (Microsoft Developer Network, 2014).

A pesar de las funcionalidades que ofrece ADO.Net, no puede ser utilizado pues es una tecnología privativa desarrollada por Microsoft, además que no tiene soporte para PostgreSQL, en este caso el gestor de base de datos utilizado en el Departamento de Tecnologías.

- **Corner Bowl Log Manager**

Es una herramienta de gestión de logs de aplicaciones en toda la red que permite a los administradores de sistemas automatizar la monitorización, consolidación de logs y requerimientos de archivado que establecen las agencias de gestión y la administración. Consolida los logs a SQL Server o MySQL, una aplicación de logs o una ubicación de la red. Monitoriza los logs en tiempo real o realiza informes programados. Por ejemplo, recibir notificación inmediata cuando cualquier usuario u ordenador intenta conectarse a la red más de 3 veces en 10 minutos. Incluye Informes de Eventos de Seguridad para ayudar con los requisitos de las auditorías. Por ejemplo, recibir informes de resumen diario o semanal con los fallos de conexión (CornerBowl Software, 2014). Escanea la red en busca de controladores, servidores, servidores de bases de datos o estaciones del dominio y las configura de una sola vez. Crea plantillas de configuración que le permiten configurar rápidamente nuevos servidores o establecer un monitor de Directorio Activo de forma que los logs de nuevos servidores se monitoricen automáticamente, se consoliden y se incluyan en los informes semanales o diarios (CornerBowl Software, 2014).

Esta herramienta no puede ser usada como solución pues no brinda soporte para PostgreSQL el gestor de base de datos que se utiliza en el Departamento de Tecnologías. (CornerBowl Software, 2014).

- **Sauxe 2.0**

Sauxe es un marco de trabajo desarrollado en la Universidad de las Ciencias Informáticas que integra en su núcleo un conjunto de tecnologías como Zend Framework 1.11, Doctrine 1.2, ExtJS 2.2. Este brinda a los desarrolladores una gran variedad de funcionalidades para facilitar el desarrollo de software. Entre ellas cuenta con un componente dedicado específicamente a la gestión de las trazas. Este componente brinda la posibilidad de registrar diferentes tipos de trazas, así como la visualización de las mismas (Cañete Pupo, 2010).

El estudio del componente de trazas de Sauxe permitió identificar algunos de los tipos de trazas que se registrarán, además posibilitó establecer relaciones entre las trazas que serán útiles en el diseño del modelo de datos. A pesar de lo anterior no es viable la utilización de este componente, pues sería necesario realizar la integración de ambos marcos de trabajo trae consigo afectaciones en el rendimiento de la solución que se desea desarrollar en cuanto a espacio en memoria y tiempos de respuestas de la aplicación.

- **AiresProxy**

El Analizador Inteligente de Registros Proxy es una aplicación web desarrollada en la UCI que proporciona reportes dinámicos de la navegación de los usuarios a través de internet. Crea reportes a partir de las trazas del servidor proxy y los usuarios pueden consultar la información de diferentes formas. Además, el sistema muestra el consumo equivalente al tamaño de los recursos accedidos mediante la WEB (AIRESProxy, 2014).

El estudio de esta herramienta posibilitó identificar criterios de búsqueda que serán incluidos en la solución que se desea desarrollar, como la búsqueda por rangos de fechas (AIRESProxy, 2014).

- **WebProfiler**

Symfony fue el primer framework en incluir una barra de depuración web. Esta muestra información para depurar las aplicaciones y acceder a todos los logs. En su versión más reciente Symfony 2 incluye un bundle llamado WebProfiler como la evolución de la su barra de depuración web. Symfony 2 genera un token único de depuración para cada página que se visualiza y al pinchar sobre este token se muestra toda la información de depuración de la página (Symfony.es, 2010).

El Profiler cuenta con cinco secciones principales:

- **Request:** contiene la información relacionada con la petición y la respuesta.
- **Exception:** si la petición ha originado una excepción, se muestra el tipo de excepción, el mensaje del servidor y toda la traza de ejecución.
- **Events:** muestra los eventos invocados durante la ejecución de la aplicación y los eventos que se han definido pero no se han llegado a ejecutar.
- **Logs:** muestra la misma información de logs que la barra de depuración original.
- **Doctrine / Propel:** muestra las consultas realizadas en la base de datos y el tiempo que ha tardado cada una.

Otra de las características del Profiler es que guarda toda su información en una base de datos SQLite llamada profiler.db y que guarda en el directorio cache/ de la aplicación. Esta base de datos posibilita consultar todo el historial de ejecución de la aplicación, lo que facilitará la depuración de los proyectos.

A pesar de toda la información que se registra en la base de datos del Profiler se decidió no utilizarlo pues no se cuenta con experiencia en el trabajo con este gestor de base de datos y el gestor utilizado por el Departamento de Tecnologías es PostgreSQL, además estos datos son guardados en un directorio de fácil acceso lo cual supone que son datos volátiles y esta base de datos no implementa ningún tipo de seguridad.

1.3. Proceso de desarrollo de software

El proceso de desarrollo de software es un conjunto estructurado de actividades y resultados asociados para desarrollar un sistema de software (Mendez, 2009). Tiene como objetivo proporcionar una guía de ejecución del proyecto que defina la secuencia de tareas que se requieren y los productos que se deben generar (Drake, 2008). Hay varios modelos a seguir para el establecimiento de un proceso para el desarrollo de software, cada uno de los cuales describe un enfoque diferente para diferentes actividades que tienen lugar durante el proceso.

1.3.1. Modelo de desarrollo de software

Un modelo de desarrollo de software define las fases por las que transita un software, las actividades a desarrollar en cada una, las disciplinas de la ingeniería de software en las que se incurre y los artefactos a generar por cada disciplina.

La solución propuesta formará parte de un proyecto desarrollado en el Departamento de Tecnologías del centro CEIGE, debido a esto se guiará por el modelo de desarrollo elaborado en dicho centro, el mismo tiene su basamento en los principios y buenas prácticas de los procesos definidos en el nivel 2 de CMMI (Capability Maturity Model Integration) (Pérez Sarduy, 2009). Este modelo se caracteriza por ser:

Centrado en la arquitectura

La arquitectura determina la línea base y los elementos de software estructurales a partir de los elementos de la arquitectura de negocio. Interviene en la gestión de cambios y diseña la evolución e integración del producto. La arquitectura orienta las prioridades del desarrollo y resuelve las necesidades tecnológicas y de soporte para el desarrollo.

Orientado a componentes

Las iteraciones son orientadas por el nivel de significancia arquitectónicas de los componentes, los mismos son abstracciones arquitectónicas de los procesos de negocio y requisitos asociados que modelan, el componente es la unidad de medición y ordenamiento de las iteraciones.

Iterativo e incremental

Las iteraciones son planificadas y coordinadas con el equipo de arquitectura, los clientes y la alta gerencia. Cada iteración constituye el desarrollo de componentes, los cuales son integrados al término de la iteración, permitiendo de esta manera la evolución incremental del producto.

Ágil y adaptable al cambio

El desarrollo de las partes formaliza solamente las características principales de la solución, priorizando los talleres y las comunicaciones entre las personas. Los clientes y funcionales están involucrados en el proyecto y poseen parte de la responsabilidad del éxito del mismo. Los cambios son conciliados semanalmente, discutidos y aprobados.

1.4. Tecnologías utilizadas en el desarrollo de la propuesta de solución

1.4.1. Framework de desarrollo

El término framework se refiere a una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Los frameworks suelen incluir soporte de programas, bibliotecas, entre otras herramientas, facilitando el desarrollo de software (Alegsa.com.ar, s.f.).

Actualmente existe una amplia gama de frameworks para desarrollar aplicaciones web que cubren un amplio número de lenguajes de programación como PHP, Ruby y Python, entre los cuales podemos destacar CodeIgniter, Symfony, Django, ZendFramework, Ruby on Rails y otros. La solución propuesta será desarrollada sobre Symfony 2 uno de los más usados hoy en día.

- **Symfony 2**

Symfony es un framework PHP para el desarrollo de aplicaciones y sitios web rápidos y seguros. Su código, y el de todos los componentes y librerías que incluye se publican bajo licencia MIT de software libre. Cuenta con un sitio online en el cual se puede encontrar una gran cantidad de documentación, también de forma gratuita (symfony.es, 2014).

Symfony 2 es su versión más reciente, se anunció por primera vez en el año 2009 y propone un cambio radical tanto en la arquitectura como en la filosofía de trabajo de versiones anteriores. Está diseñado para explotar las potencialidades de PHP 5.3, lo cual supone mejoras en el rendimiento del framework. Su arquitectura está totalmente desacoplada, esto permite eliminar aquellos componentes que no necesitamos para el desarrollo de nuestro proyecto (symfony.es, 2014).

- **ExtJS 4.0**

Es una librería Java Script open-source de alto rendimiento para la creación y desarrollo de aplicaciones web dinámicas. Provee interfaces gráficas de usuario que brindan experiencias parecidas o iguales a las que se tienen con aplicaciones de escritorio. Permite la creación de aplicaciones complejas utilizando componentes predefinidos. Es extensible para la gran mayoría de los navegadores, evitando el tedioso problema de validar el código para cada uno de estos. Entre sus principales ventajas se encuentra el balance entre Cliente-Servidor, distribuyendo la carga de procesamiento en el último, y este al tener menor carga, maneja los clientes de manera más eficiente. La comunicación asíncrona permite el intercambio de información con el servidor sin necesidad de pedirle una acción al usuario, dando la libertad de cargar la información sin que este lo note (Sencha, 2014).

- **Twig 1.13**

Es un motor de plantillas para PHP cuyo objetivo es ofrecer una alternativa flexible, potente y segura. Ha sido creado por los desarrolladores del marco de trabajo Symfony. Su sintaxis se origina a partir de Jinja y plantillas de Django. Es un producto de código abierto se distribuye bajo licencia BSD.Symfony2 viene con soporte incluido para Twig como su motor de plantillas por defecto (Pacheco, 2011).

1.4.2. ORM

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational Mapping, o sus siglas ORM) es una técnica de programación que posibilita el acceso a una base de datos relacional desde una aplicación desarrollada según los principios de la programación orientada a objetos, haciendo uso de una interfaz que traduce los objetos en registros de las tablas y viceversa (Fernández Díaz, 2012). Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

- **ORM Doctrine 2.0**

Doctrine 2 es un asignador-objeto-relacional (ORM) para PHP 5.3.0 o superior que proporciona persistencia transparente de objetos PHP situado en la parte superior de una poderosa capa de abstracción de base de datos (DBAL por DataBase Abstraction Layer) (Doctrine Project Team, 2011).

Una de las características clave de Doctrine es la opción de escribir las consultas de base de datos en un dialecto SQL propio orientado a objetos llamado Lenguaje de Consulta Doctrine (DQL

por Doctrine Query Language), inspirado en Hibernates HQL. Además DQL difiere ligeramente de SQL en que abstrae considerablemente la asignación entre las filas de la base de datos y objetos, permitiendo a los desarrolladores escribir poderosas consultas de una manera sencilla y flexible (Doctrine Project Team, 2011).

Dentro de las principales ventajas que aporta el ORM es la reutilización, permitiendo llamar a los métodos de un objeto de datos desde varias partes de la aplicación e incluso desde diferentes aplicaciones. Además la utilización de objetos en vez de registros y de clases en vez de tablas, tiene otra ventaja: permite añadir métodos de acceso en los objetos que no tienen relación directa con una tabla (LIBROSWEB.es, 2010).

1.4.3. Lenguajes de programación.

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana (Mark, 2010).

- **PHP (acrónimo de *"PHP: Hypertext Pre-processor"*)**

PHP es un lenguaje de programación web de "código abierto" interpretado, de alto nivel, embebido en páginas HTML y ejecutado en el servidor (Achour, y otros, 2006). Este lenguaje de programación nace en la década de los 90´s y actualmente es uno de los lenguajes de programación más importantes y populares. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno.

A grandes rasgos la lógica de PHP es la siguiente: cuando el cliente hace una petición al servidor para que le envíe una página web, el servidor ejecuta el intérprete de PHP. Éste procesa el script solicitado que generará el contenido de manera dinámica (por ejemplo obteniendo información de una base de datos). El resultado es enviado por el intérprete al servidor, quien a su vez se lo envía al cliente. Este lenguaje además permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, PostgreSQL, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite (Rincón Carrera, 2011).

- **JavaScript**

JavaScript es un lenguaje de programación dinámico que soporta construcción de objetos basado en prototipos. La sintaxis básica es similar a Java y C++ con la intención de reducir el número de

nuevos conceptos necesarios para aprender el lenguaje. Las construcciones del lenguaje, tales como sentencias if, y bucles for y while, y bloques switch y try... catch funcionan de la misma manera que en estos lenguajes (o casi).

JavaScript puede funcionar como lenguaje procedimental y como lenguaje orientado a objetos. Los objetos se crean programáticamente añadiendo métodos y propiedades a lo que de otra forma serían objetos vacíos **en tiempo de ejecución**, en contraposición a las definiciones sintácticas de clases comunes en los lenguajes compilados como C++ y Java (Mozilla Developer Network, 2011).

1.5. Herramientas utilizadas para el desarrollo de la propuesta de solución

1.5.1. Herramientas Case

Una herramienta de ingeniería de software asistida por computadora (CASE de sus siglas en inglés) es un software que soporta varias actividades de la ingeniería de software dentro del proceso de desarrollo de software. Se utilizan principalmente para la confección de diagramas y modelos que soportan todas las fases del ciclo de vida del software principalmente el análisis y diseño. Proveen también funcionalidades para la generación automática de código. Lo anterior favorece al desarrollo colaborativo y al mantenimiento (Tomar, 2011). Algunos de los procesos que soportan las herramientas CASE son:

- Creación de flujos de datos y modelos de entidad de relación.
- Establecimiento de relaciones entre los requerimientos y modelos.
- Desarrollo de diseños de alto nivel.
- Construcción de diagramas de procesos.
- Creación de casos de pruebas.

Como herramientas CASE para facilitar el modelado, la construcción de diagramas y la generación automática de código en la presente investigación se decide utilizar Visual Paradigm for UML. A continuación se describe brevemente esta herramienta.

- **Visual Paradigm 8.0**

Visual Paradigm es una herramienta de diseño UML libre y profesional, diseñado para contribuir al desarrollo de software. Soporta los principales estándares como UML, BPMN y XML. Ofrece un completo conjunto de herramientas a los equipos de desarrollo de software, necesarios para

la captura de requisitos, la planificación de software, la planificación de pruebas, el modelado de clases y el modelado de datos (Morales, Marrero, & Oliva, 2013).

- **Lenguaje de Modelado UML**

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios (Schmuller, 2001).

1.5.2. Sistema Gestor de Base de Datos

Un sistema gestor de base de datos (SGBD) es básicamente un sistema computarizado cuya finalidad general es almacenar información y permitir a los usuarios recuperar y actualizar esa información a través de peticiones o consultas. La información puede ser cualquier dato que resulte importante para el individuo u organización; en otras palabras, todo lo necesario para auxiliar el proceso general de su administración. En estos sistemas se proporciona un conjunto coordinado de programas, procedimientos y lenguajes que permiten a los distintos usuarios realizar sus tareas habituales con los datos, garantizando además la seguridad de los mismos (Sánchez Asenjo, 2013). Algunos de los SGBD libres son SQLite, DB2 Express-C, Apache Derby, MySQL, PostgreSQL, entre otros. Para el manejo de los datos en el sistema propuesto se utilizará como SGBD PostgreSQL.

- **PostgreSQL 9.2**

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales. (PostgreSQL-es, 2014)

Incluye características como la herencia, valores no atómicos (atributos basados en vectores y conjuntos), funciones, disparadores, entre otras. Es altamente extensible, permitiendo el uso de operadores, funciones y tipos de datos definidos por el usuario. Soporta la integridad referencial garantizando la integridad de los datos en la base de datos (PostgreSQL-es, 2014).

PostgreSQL permite realizar múltiples conexiones desde procesos clientes, existiendo un proceso maestro en el servidor que siempre se ejecuta y está a la espera de nuevas conexiones clientes, de forma tal que cuando alguien se conecta, se inicia un nuevo proceso, asegurando

que la nueva conexión no necesite del proceso postgres original, por lo que una de sus principales características es la alta concurrencia (PostgreSQL-es, 2014).

- **PgAdmin 3**

PgAdmin es la plataforma de administración y desarrollo más popular de código abierto para PostgreSQL, la base de datos más avanzado de código abierto en el mundo. La aplicación puede utilizarse en Linux, FreeBSD, Solaris, Mac OSX y Windows para administrar PostgreSQL 7.3 y superiores, así como las versiones comerciales y derivados de PostgreSQL como Postgres Plus Advanced Server y base de datos Greenplum (pgAdmin, 2014).

Está diseñado para responder a las necesidades de todos los usuarios de escribir consultas SQL sencillas para desarrollar bases de datos complejas. La interfaz gráfica es compatible con todas las características de PostgreSQL y facilita la administración. La aplicación también incluye un editor de resaltado de sintaxis SQL, un editor de código del lado del servidor, un agente de planificación de tareas de SQL / batch / shell, el apoyo a la Slonyl motor de replicación y mucho más. Conexión con el servidor se puede hacer a través de TCP / IP o Unix Domain hembra (en plataformas * nix), y puede ser encriptado SSL para la seguridad. No se requieren controladores adicionales para comunicarse con el servidor de base de datos (pgAdmin, 2014).

1.5.3. Entorno de Desarrollo Integrado.

Un Entorno de Desarrollo Integrado (IDE de sus siglas en inglés) brinda todas las herramientas de programación dentro de un único espacio. Antiguamente los programadores tenían que editar ficheros, salvarlos fuera, correr el compilador, construir la aplicación y ejecutarla a través de un depurador. Actualmente un IDE integra en una única aplicación: editor, compilador, el depurador y herramientas para la administración de los proyectos. Todas estas posibilidades favorecen el aumento de la productividad del programador (Bolton, 2014). Por sus ventajas, popularidad y facilidades además de la experiencia de los autores de esta investigación en el trabajo con este Entorno de Desarrollo Integrado, se decide utilizar NetBeans IDE para soportar la implementación de la solución.

- **Netbeans IDE 7.4**

NetBeans es un proyecto exitoso de código abierto con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos (NetBeans, 2014).

NetBeans IDE es un entorno de desarrollo - una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para otros lenguajes como PHP, Groovy, HTML, C#, C++, entre otros. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso. El código fuente está disponible para su reutilización de acuerdo con la Common Development and Distribution License (CDDL) v1.0 and the GNU General Public License (GPL) v2 (NetBeans, 2014).

- **Mozilla Firefox 25.0**

Mozilla Firefox es un navegador web libre y de código abierto desarrollado por Mozilla, una comunidad global que trabaja junta para mantener una Web Abierta, pública y accesible para todos (Mozilla, 2014).

Su primera versión vio la luz el 9 de noviembre de 2004 y desde allá hasta la fecha, Firefox ha revolucionado la forma de pensar y mantenido la innovación en la web para llevar a sus usuarios una mejor experiencia. Entre sus méritos recae el orgullo de ser uno de los proyectos de Software Libre más importantes del mundo y el récord Guinness al software más descargado en 24 horas con 8 002 530 de descargas, batido el 17 de junio de 2008 (Mozilla, 2014).

Cuenta con alucinantes características entre las que se encuentran: navegación por pestañas, corrector ortográfico, búsqueda progresiva, marcadores dinámicos, un administrador de descargas, navegación privada, aceleración mediante GPU y muchas otras más. También mediante complementos puedes extenderlo añadiéndole funcionalidades que no trae "de fábrica". Además, es 100% compatible con estándares web, manteniendo así la elección e innovación en la web (Mozilla, 2014).

1.5.4. Control de versiones

El control de versiones es la capacidad de recordar todos los cambios que se hacen tanto en la estructura de carpetas como en el contenido de los ficheros cuando más de una persona trabaja sobre los mismos archivos y es necesario mantener el control sobre los cambios que se realizan. Subversion es una herramienta para el control de versiones de ficheros electrónicos, como son el software y la documentación. Se basa en un repositorio central que actúa como un servidor de ficheros. Existen herramientas denominadas clientes que se configuran para conectarse a dichos servidores, dos de las más comunes son TortoiseSVN y RapidSVN (Serradilla, 2007). Para el trabajo con el control de versiones durante el desarrollo del sistema se decidió utilizar TortoiseSVN por la experiencia de los autores trabajando con la misma.

- **TortoiseSVN 1.7.12**

TortoiseSVN es un cliente gratuito de código abierto para el sistema de control de versiones Apache™ Subversion. Administra archivos y directorios a lo largo del tiempo. Los archivos se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de archivos ordinario, con la excepción de que recuerda todos los cambios que se hayan hecho a sus archivos y directorios. Esto le permite al usuario recuperar versiones antiguas de sus archivos y examinar la historia de cómo y cuándo cambiaron sus datos, y quién hizo el cambio. Esta es la razón por la que mucha gente piensa en Subversion, y los sistemas de control de versiones en general, como una especie de "máquinas del tiempo" (TortoiseSVN, 2014).

TortoiseSVN se integra perfectamente con la consola de Windows (por ejemplo, el explorador). Esto significa que puede seguir trabajando con las herramientas que ya conoce. Y que no tiene que cambiar a una aplicación diferente cada vez que necesite las funciones del control de versiones (TortoiseSVN, 2014).

Los menús contextuales de TortoiseSVN también funcionan en otros administradores de archivos, y en el diálogo Archivo/Abrir que es común en la mayoría de aplicaciones estándar de Windows. Sin embargo, debe tener en cuenta que TortoiseSVN está desarrollado con la intención de ser una extensión del Explorador de Windows. Por este motivo, puede que en otras aplicaciones la integración no sea tan completa, y que por ejemplo, los íconos superpuestos no se muestren (TortoiseSVN, 2014).

1.6. Requisitos de software

Los requisitos de software expresan las necesidades y apremios colocados en un producto de software que contribuyen a la solución de un cierto problema del mundo real.

La captura de requisitos de software es una de las actividades fundamentales que se realiza en el proceso de desarrollo de software. Los mismos se dividen en:

Requisitos funcionales: son capacidades o condiciones que el sistema debe cumplir.

Requisitos no funcionales: son propiedades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable (IEEE, 2004).

1.7. Técnicas de captura de requisitos

La captura de requisitos es la actividad mediante la que se extraen las necesidades del sistema, es el comienzo de cada ciclo de desarrollo. Las técnicas de captura son las que posibilitan que la extracción de los requisitos sea efectiva y sirven de base para verificar si se alcanzaron los objetivos establecidos en la investigación. Seguidamente se describen las técnicas que se utilizan:

Entrevistas: Es un método clásico que posibilita tomar conocimiento del problema y comprender los objetivos de la solución propuesta. Mediante esta técnica el equipo de desarrollo se acerca al problema de una forma natural consultando al cliente (IEEE, 2004).

Sistemas Existentes: Consiste en analizar y estudiar los sistemas existentes que estén relacionados con la solución del sistema que va a ser construida, analizándose las interfaces de usuarios y el comportamiento que el mismo produce.

1.8. Técnicas de validación de requisitos

La validación de requisitos permite demostrar que la descripción de requisitos define realmente la solución que el usuario necesita. Su objetivo es verificar todos los requisitos obtenidos para asegurarse que presentan una descripción correcta del sistema a implementar. A continuación se detallan las técnicas que se emplearán para validar los requisitos funcionales:

Revisiones de requisitos: Consiste en la asignación de un grupo de personas para revisar los documentos de descripción de requisitos en busca de errores, falta de claridad, etc. De esta forma se puede validar la correcta interpretación de la información transmitida (IEEE, 2004).

Prototipado: Construir prototipos es un medio para validar la interpretación de los requisitos de software, puede facilitar en gran medida dicha interpretación y brindar una explicación útil de porque son incorrectas (IEEE, 2004).

1.9. Patrones de diseño

Un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlos en nuevas situaciones (Larman, 1999).

Los patrones de diseño (del inglés design patterns) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es una solución a un problema del diseño (Gamma).

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

1.9.1. Patrones Grasp

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos (Larman, 1999). Existen nueve patrones GRASP pero a continuación se mencionan y describen los que se utilizaron durante el desarrollo de la investigación:

Experto: El patrón experto en información es el principio básico de asignación de responsabilidades. 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).

Creador: El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, o usa directamente las instancias creadas del objeto.

Controlador: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar

la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento.

Alta Cohesión: Este patrón determina, que la información almacenada en una clase debe ser coherente y estar relacionada con esta, en mayor medida y enfocada en sus responsabilidades. Al realizar un diseño donde las clases del componente mantengan una alta cohesión como por ejemplo las clases controladoras, es posible ganar en claridad y facilidad a la hora de entender el diseño, además de simplificar el mantenimiento y soportar mayor capacidad de reutilización.

Bajo acoplamiento: Consiste en tener las clases lo menos relacionadas entre sí, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás, potenciando la reutilización y disminuyendo la dependencia entre las clases.

1.9.2. Patrones Gof

Son patrones de diseño publicados en el libro Design Patterns: Elements of Reusable Object-Oriented Software por Gamma, Helm, Jonson y Vlissides conocidos mundialmente por Gang of Four o Pandilla de los cuatro. Se clasifican en creacionales, estructurales y de comportamiento (Gamma).

- Los patrones creacionales se encargan de la creación de los objetos ayudando a que el sistema sea independiente de la creación, composición y representación de los objetos.
- Los patrones estructurales son los encargados de cómo las clases y objetos están compuestos para formar estructuras más grandes. Los patrones estructurales usan la herencia para componer interfaces u objetos en tiempo de ejecución.
- Los patrones de comportamiento plantean algoritmos y la asignación de responsabilidades entre objetos. Estos patrones no sólo describen clases y objetos sino también describen la comunicación entre ellos.

- **Observer**

Diferentes tipos de objetos suscriptores están interesados en el cambio de estado o eventos de un objeto emisor, y quieren reaccionar cada uno a su manera cuando el emisor genere un evento. Además, el emisor quiere mantener bajo acoplamiento con los suscriptores (Larman, 1999).

- **Singleton**

Limita el número de instancias de un objeto a uno. Los clientes que quieran usar dicho objeto compartirán la única instancia existente (Ramirez, 2004).

1.10. Arquitectura de software

La arquitectura representa un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de framework en los que se pueden integrar componentes (Reynoso, 2004).

Paul Clements, plantea que la arquitectura de software es, a grandes rasgos, una vista general del sistema que incluye los componentes principales, la conducta de estos componentes con el resto del sistema y las formas en que deben interactuar y coordinarse para alcanzar la misión del sistema (Clements, 1996).

Según Pressman, la arquitectura de software es la representación que capacita al ingeniero del software para: analizar la efectividad del diseño para la consecución de los requisitos fijados, considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil, y reducir los riesgos asociados a la construcción del software. (Pressman, 2005).

Sin embargo, para este trabajo se utilizará la definición oficial establecida por la IEEE: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.”

1.11. Conclusiones parciales

A lo largo del capítulo se definieron una serie de conceptos necesarios para el entendimiento del problema tratado en el presente trabajo y se realizó un estudio de las herramientas para el registro y monitoreo de las trazas.

La investigación realizada demuestra que no existe una herramienta de registro de trazas que se adapte completamente a las características del marco de trabajo Symfony 2. Por un lado la gran mayoría de las herramientas existentes están desarrolladas para ser utilizadas en bases de datos Oracle y SQL Server, y el gestor de base de datos requerido es PostgreSQL. Otra de las

limitaciones encontradas es el elevado coste por concepto de licencias, que atenta contra la política del centro de utilizar tecnologías libres.

Por lo antes mencionado se determinó que es necesario el desarrollo de una solución que se pueda integrar al marco de trabajo Symfony 2, que permita el registro organizado y abstracto de las trazas. Para ello se definió una metodología para el proceso de desarrollo del software, y se propusieron las herramientas y tecnologías que se utilizarán para llevar a cabo el desarrollo de la solución.

2. CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA

El siguiente capítulo recoge los resultados obtenidos durante el proceso de desarrollo de la solución, así como algunos de los artefactos generados por el mismo. Primeramente se elabora un modelo conceptual para lograr un entendimiento de lo que se quiere desarrollar, y luego se especifican y describen los requisitos funcionales de la solución. Posteriormente se realiza una descripción del diseño elaborado por los autores para alcanzar las metas trazadas.

2.1. Descripción de la solución propuesta

El sistema informático a desarrollar sobre Symfony 2 y utilizando el marco de trabajo Extjs para la capa de presentación, debe permitir la gestión de las trazas generadas en las aplicaciones web desarrolladas sobre dicha tecnología. Para ello el nuevo módulo llamado Trazas será encargado de llevar cabo esta labor. Entre las responsabilidades principales de este módulo está el registro de las trazas de acción, excepción, datos y rendimiento; así como brindar una interfaz que visualice las trazas registradas y permita hacer filtros por varios criterios.

Este trabajo investigativo persigue que el módulo Trazas pueda ser reutilizable y se integre con otros módulos, como Seguridad y Estructura y Composición, mediante la implementación de interfaces que brinden o soliciten servicios específicos.

Symfony 2 proporciona facilidades para el trabajo, y acentúa una arquitectura que promueve el código reutilizable y disociado. La estructura de directorios de una aplicación Symfony 2 es la siguiente:

- app\: Configuración de la aplicación.
- src\: El código PHP del proyecto.
- vendor\: Las dependencias de terceros.
- web\: El directorio raíz del servidor web.

La configuración de todo el proyecto se encuentra en el directorio app\. En src se encuentra el código de la aplicación, la cual estará formada por módulos denominados bundles. En Symfony 2 existe el concepto de bundle, el cual es un conjunto de archivos que implementan una única funcionalidad. La configuración de un bundle se puede realizar a través del archivo config.yml en la carpeta config\ del proyecto, o a través del archivo services.yml que se encuentra en la carpeta config\ dentro del bundle; de cualquier modo en esta configuración solo se explicitan los servicios que provee un bundle, pudiéndose consumir dichos servicios desde cualquier parte del sistema.

Un servicio es cualquier objeto PHP que realiza alguna tarea “global”, un objeto creado para un propósito determinado. El servicio se utiliza en el sistema siempre que se necesite la funcionalidad específica que este proporciona.

Al declarar un servicio en alguno de los archivos de configuración también se detallan los parámetros que este debe recibir, así como la ubicación de la clase que lo implementa. El bundle Trazas estará dentro de la carpeta src\ y debe existir un fichero composer.json en la raíz de la aplicación que contenga sus metadatos, servicios y dependencias.

Este fichero tiene la siguiente estructura:

{“services”: [

“identificador servicio”:

{“interface”: “namespace relativa a la interfaz”,

“impl:” “namespace relativa a la implementación”}

], “dependencies”: [“identificador dependencia”:

{“interface”: “namespace relativa a la interfaz”,

“optional”: “valor booleano”,

“use”: “nombre componente que resuelve”

“versión”: “X.X.X”}]}

En las dependencias, el uso de las propiedades *optional*, *use*, y *versión* no es obligatorio. La primera si está en false, garantiza que si no se resuelve esa dependencia el componente no funciona. La segunda es para especificar qué componente y la tercera qué versión del componente puede resolver la dependencia. La versión puede no escribirse, y también puede usar comodines por ejemplo: >=1.0.*.

2.2. Modelado del negocio

Antes de identificar los requisitos funcionales es necesario conocer el dominio del problema de la investigación y los contextos organizacional y operacional, es decir la situación actual. El modelado del negocio constituye una actividad fundamental para comprender las actividades que se llevan a cabo dentro de la organización, conocer las necesidades del cliente y tener garantías de que el software desarrollado va a cumplir su propósito (González Obregón, 2012).

2.3. Modelo conceptual

El Modelo conceptual es una representación de conceptos del mundo real. Este artefacto se crea con el objetivo de aumentar la comprensión del problema y contribuir a esclarecer la terminología usada, comunica a los involucrados cuáles son los términos importantes y cómo se relacionan entre sí. Rigiéndose por lo establecido en la disciplina Modelado de negocio, del modelo de desarrollo de software, se identificaron 5 conceptos principales del módulo Trazas y se elaboró el Modelo conceptual del mismo.

El siguiente modelo de dominio muestra el marco de trabajo Symfony 2, el cual cuenta con varios componentes (Estructura y Composición, Seguridad, Portal y Trazas) y las relaciones entre ellos. Estos a su vez tienen asociados procesos que para desarrollarse acceden a diferentes funcionalidades. Dichas funcionalidades contienen una serie de acciones que pueden ser ejecutadas, cuando se ejecuta una de estas se desencadenan una serie de eventos en el sistema, los cuales se registraran como trazas.

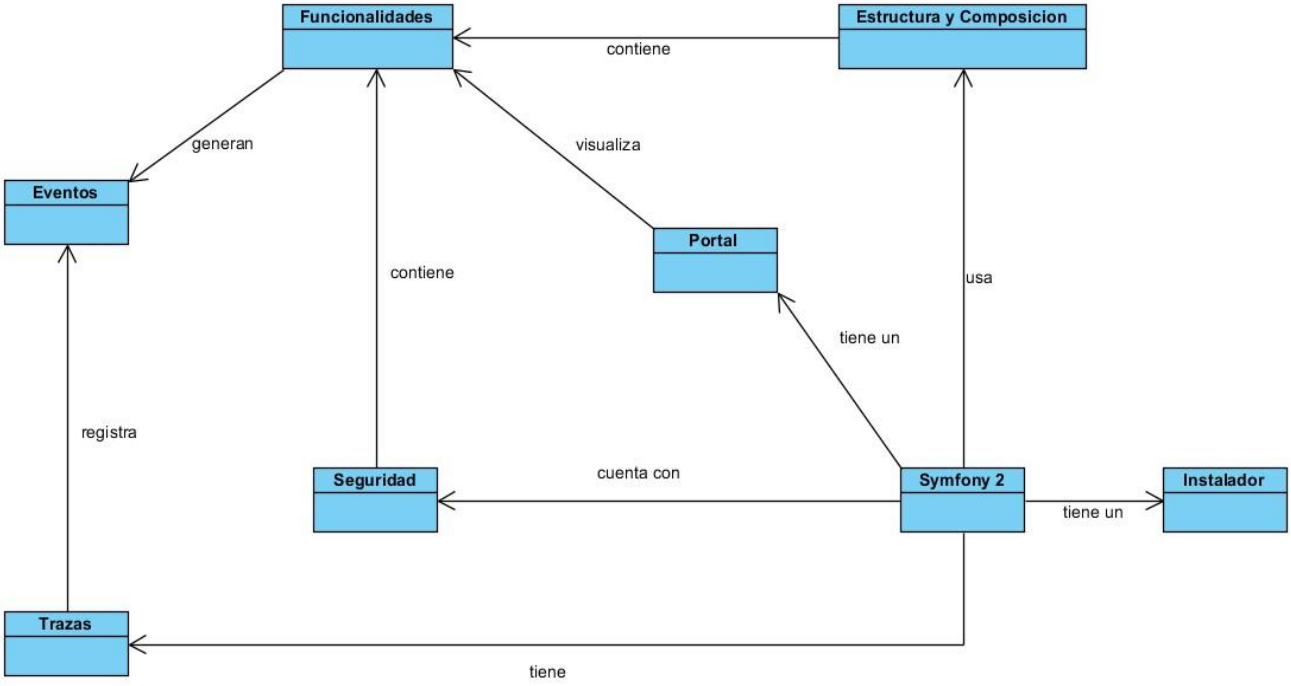

Figura 1: Modelo de dominio.

A continuación se presenta el modelo conceptual del módulo Trazas, en este se ejemplifican los diferentes tipos de trazas que se gestionan y las relaciones entre las mismas.

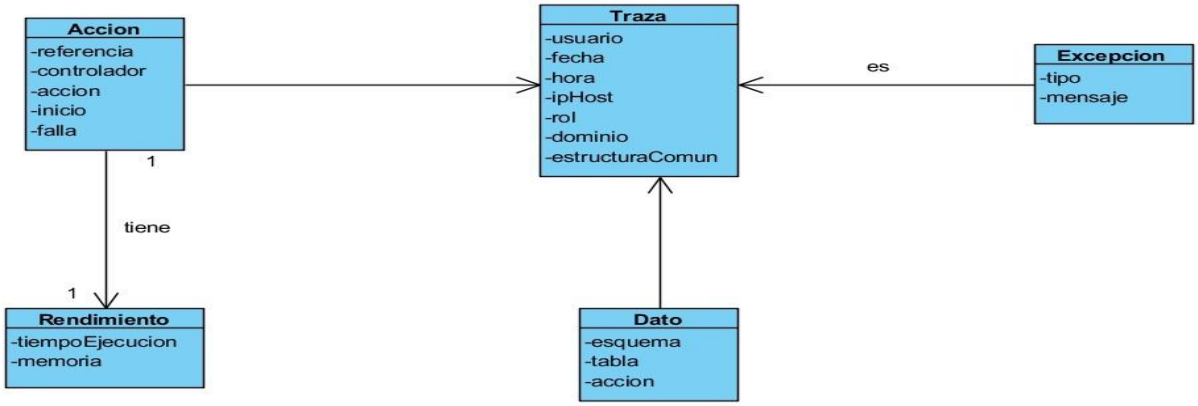

Figura 2: Mapa conceptual del módulo Trazas

2.4. Requisitos de software

Con la identificación y la descripción de los requisitos de software se pretende establecer y mantener un acuerdo con los clientes y otros involucrados en lo que el sistema debe hacer. Permite definir la interfaz de usuario del sistema enfocándose en las necesidades y aspiraciones de los usuarios.

2.4.1. Requisitos funcionales

Mediante las técnicas de captura de requisitos se entrevistaron a desarrolladores y analistas del centro CEIGE, principalmente del Dpto. de Tecnologías, para constatar en qué consiste el funcionamiento del módulo de trazas que este posee y luego modelar el diseño de la solución. Se estudiaron herramientas relacionadas con el objeto de estudio, con el objetivo de identificar potenciales funcionalidades y aspectos nuevos que pudieran concebirse en el nuevo sistema. Finalmente se obtuvo un total de 5 requisitos funcionales, los cuales se muestran a continuación.

- RF1. Registrar trazas de datos.
- RF2. Registrar trazas de acción.
- RF3. Registrar trazas de excepción.
- RF4. Registrar trazas de rendimiento.
- RF5. Listar trazas.

2.4.2. Especificación de requisitos funcionales

En las especificaciones de requisitos son plasmadas las características y condiciones precisas que debe cumplir cada requisito funcional. Seguidamente se presenta la especificación de algunos de los requisitos funcionales, los restantes se pueden consultar en los anexos:

Tabla 1: Especificación del requisito Registrar Traza de Acción.

Precondiciones	Un usuario inicia sesión en el sistema		
Flujo de eventos			
Flujo básico Registrar traza de acción			
1	El usuario realiza alguna acción en el sistema.		
2	Se capturan los datos de la traza: -Fecha -Hora -Usuario -Ip host -Rol -Dominio	 -Estructura común -Referencia -Controlador -Acción -Inicio -Falla	
3	Concluye el requisito		
Post-condiciones			
1	Se persisten los datos en la base de datos		
Flujos alternativos			
Flujo alternativo <<Nº Evento>>.<<letra iniciando por la a>> <Condición que dio lugar a la extensión>			
1			
Post-condiciones			
1			
Validaciones			
1	Se validan los datos según lo establecido en el Modelo conceptual CEIGE_Modelo_conceptual_Trazas.		
Conceptos	**Traza de acción**	Utilizados internamente: -Fecha -Hora -Usuario	 -Estructura común -Referencia -Controlador

		-Ip host -Rol -Dominio	-Acción -Inicio -Falla
Requisitos especiales	N/A		
Asuntos pendientes	N/A		

Precondiciones	-El usuario debe autenticarse en el sistema. -Existen trazas registradas en el sistema.
Flujo de eventos	
Flujo básico Listar Trazas	
1	El usuario accede a la funcionalidad Listar Trazas.
2	Selecciona el tipo de traza que desea listar.
3	Si el tipo de traza seleccionado es “Acción”.
4	Se muestra las trazas de acción registradas en el sistema especificando los siguientes campos: -Fecha -Estructura común -Hora -Referencia -Usuario -Controlador -Ip host -Acción -Rol -Inicio -Dominio -Falla
5	Si no se desea establecer un rango de fechas.
6	Concluye el requisito.
Post-condiciones	
1	
Flujos alternativos	
Flujo alternativo 3.a Si el tipo de traza seleccionada es “Datos”	
1	Se muestra las trazas de datos registradas en el sistema especificando los siguientes campos:

	-Fecha	-Dominio
	-Hora	-Estructura común
	-Usuario	-Esquema
	-Ip host	-Tabla
	-Rol	-Acción
2	Volver al paso 5 del flujo básico	
Post-condiciones		
1		
Flujo alternativo 3.b Si el tipo de traza seleccionada es "Rendimiento"		
1	Se muestra las trazas de rendimiento registradas en el sistema especificando los siguientes campos:	
	-Fecha	-Dominio
	-Hora	-Estructura común
	-Usuario	-Tiempo de Ejecución
	-Ip host	-Memoria
	-Rol	
2	Volver al paso 5 del flujo básico	
Post-condiciones		
1		
Flujo alternativo 3.c Si el tipo de traza seleccionada es "Excepción"		
	Se muestra las trazas de excepción registradas en el sistema especificando los siguientes campos:	
	-Fecha	-Dominio
	-Hora	-Estructura común
	-Usuario	-Tipo
	-Ip host	-Mensaje
	-Rol	
1	Volver al paso 5 del flujo básico	
Post-condiciones		
1		
Flujo alternativo 5.a Si se establece un rango de fechas		

1	-Se introducen las fechas.	
2	-El sistema valida que el rango de fechas sea correcto. (Ver validación 1).	
3	-Si los valores introducidos son correctos el sistema reduce el listado al rango de fechas establecido.	
4	-Volver al paso 6 del flujo básico.	
Validaciones		
1	Se validan los datos según lo establecido en el Modelo Conceptual CEIGE_Modelo_conceptual_Trazas.	
Conceptos	**Traza de acción**	Utilizados internamente: -Fecha -Hora -Usuario -Ip host -Rol -Dominio -Estructura común -Referencia -Controlador -Acción -Inicio -Falla
	Traza de datos	Utilizados internamente: -Fecha -Hora -Usuario -Ip host -Rol -Dominio -Estructura común -Referencia -Controlador -Esquema -Tabla -Acción
	Traza de rendimiento	Utilizados internamente: -Fecha -Hora -Usuario -Ip host -Rol -Dominio -Estructura común -Referencia -Controlador -Acción -Inicio -Falla
	Traza de excepción	Utilizados internamente: -Fecha -Hora -Dominio

		-Usuario -Ip host -Rol	-Estructura común -Tipo -Mensaje
Requisitos especiales	N/A		
Asuntos pendientes	N/A		

2.4.3. Requisitos no funcionales

Los resultados del presente trabajo investigativo formarán parte de un nuevo marco de trabajo desarrollado en Symfony 2 que se utilizará en el Departamento de Tecnologías del centro CEIGE, por lo tanto los requisitos no funcionales de la solución propuesta se acogerán a los establecidos por el centro, inmediatamente se describen algunos de los más significativos.

Usabilidad: El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Seguridad: Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos. Verificación sobre las acciones irreversibles como las eliminaciones.

Soporte: Los desarrolladores deben utilizar para el desarrollo de la solución el estándar de codificación elaborado en el Departamento de Tecnología del CEIGE.

Portabilidad: El sistema debe ser multiplataforma.

2.5. Modelado del diseño

El modelo de diseño describe la realización física de los procesos, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de programación, tienen impacto en el sistema a considerar, siendo una entrada fundamental para las actividades de implementación. Este modelo se puede utilizar para visualizar la implementación y para soportar las técnicas de programación gráfica de la aplicación. Para una mejor calidad del diseño fueron aplicados patrones de diseño durante la realización de los diagramas de clases permitiendo asignar las responsabilidades a los objetos y diseñar la colaboración entre ellos. Dentro de este epígrafe se verán los elementos que se tuvieron en cuenta durante el diseño de la solución.

2.5.1. Patrones de diseño

Los patrones de diseño se aplican para establecer un lenguaje común entre los programadores, contribuir a la reutilización, ahorrar tiempo en la implementación y obtener un producto con calidad. Entre sus características debe incluir la habilidad de ser reutilizable y aplicable a diferentes problemas de diseño en distintas circunstancias. Seguidamente se describen como se manifiesta la utilización de los patrones de diseño para este trabajo investigativo.

2.5.1.1. Patrones GRASP

Symfony 2 evidencia el uso de varios patrones de diseño que este trae incluidos por defecto en su arquitectura, además de estar concebidos de tal manera que obliga al programador a aplicarlos. Entre ellos tenemos:

- **Experto**: Es uno de los patrones que más se utiliza cuando se trabaja con Symfony 2, con la inclusión de Doctrine para mapear la base de datos. Symfony 2 utiliza este ORM para implementar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad y contienen la información necesaria de la tabla que representan.
- **Creador**: En las clases controladoras del bundle de Trazas se encuentran las funcionalidades con el sufijo “action”, aquí se encuentran las acciones definidas para el sistema y se ejecutan en cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que funcionalidades son creadoras de dichas entidades.
- **Controlador**: Todas las peticiones web son manipuladas por un controlador frontal, ejemplo de esto son las clases “app.php” y “app_dev.php”, que es el punto de entrada único de toda la aplicación en un entorno determinado. Este patrón se evidencia en las clases controladoras del sistema. En Symfony 2 hay una capa específicamente para los controladores, que son el núcleo del framework, aquí cada clase tiene su responsabilidad y es única, hay controladores que se encargan de la seguridad del sistema trabajando con ficheros “yml” de configuración.
- **Alta Cohesión**: Una de las características principales del marco de trabajo Symfony 2 es la organización del trabajo en cuanto a la estructura del proyecto. Realizando un diseño donde las clases mantengan una alta cohesión, permite que el software sea flexible a

cambios sustanciales con efecto mínimo. Se emplea en las clases controladoras ya que fueron asignadas responsabilidades a las clases de forma tal que la cohesión siguiera siendo alta, o sea, cada clase se encargará de realizar solamente las funciones que estén en correspondencia con la responsabilidad que posea.

- **Bajo acoplamiento:** Este patrón se evidencia dentro del marco de trabajo Symfony 2 en las clases que implementan la lógica del negocio y de acceso a datos que se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista , lo que proporciona que la dependencia en este caso sea baja. Como existe poca dependencia entre esas clases se permite una mayor reutilización.

2.5.1.2. Patrones Gof

- **Observer:** Symfony 2 utiliza este patrón de diseño para la gestión de sus eventos. Notifica diversos eventos en los momentos más importantes que se producen durante la ejecución de una petición. El mismo se evidencia en las clases de tipo listener, creadas para capturar la información necesaria para registrar las trazas a partir de los eventos que se producen en el sistema.
- **Singleton:** Este patrón se implementó en la clase AccionListener para escuchar dos eventos diferentes que se generan en el sistema y se hace necesario que ambos accedan a una única instancia de dicha clase.

2.5.2. Arquitectura orientada a componentes

Es un estilo arquitectónico impuesto al diseño del sistema, el objetivo es establecer una estructura para todos los componentes del mismo, ya que describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Está enfocado a la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas y proporcionen los servicios requeridos en la aplicación. En esta arquitectura la interfaz es el elemento básico de interconectividad, por lo que cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. La modularidad y la reusabilidad son características muy relevantes de una arquitectura basada en componentes (Pressman, 2005).

Este estilo arquitectónico trae consigo una serie de beneficios como la facilidad de instalación, o sea, cuando una nueva versión de la aplicación esté disponible puede reemplazarse la versión existente sin impactar otros componentes o el sistema como un todo. También permite la

reusabilidad de los componentes entre diversas aplicaciones y por consiguiente, reducir los costos de desarrollo y mantenimiento.

Existen varias definiciones del término componente, para este trabajo se utilizará la enunciada en el libro Ingeniería de Software: Un enfoque práctico-6ta edición, de Roger Pressman, la cual plantea que:

"Un componente es una parte modular desplegable y remplazable de un sistema que encapsula la implementación y expone un conjunto de interfaces, que permiten que las clases se comuniquen y colaboren con otras clases de diseño."

Los componentes de software deben cumplir ciertas características que a continuación de describen:

Identificable: Tener una identificación clara consistente que facilite su catalogación y búsqueda en repositorios de componentes.

Accesible sólo a través de su interfaz: Debe exponer al público únicamente el conjunto de operaciones que lo caracteriza y ocultar los detalles de su implementación.

Servicios invariantes: Las operaciones que ofrece a través de su interfaz no deben variar. La implementación de estos servicios puede ser modificada, pero no debe afectar la interfaz.

Documentado: Debe tener una documentación adecuada que facilite su evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte.

2.5.3. Patrón de arquitectura

Un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución (Facultad de Ciencias, UNAM, 2002).

Symfony 2 basa su funcionamiento interno en el estilo Modelo-Vista-Controlador (MVC), uno de los más usados por los frameworks web (Eguiluz, 2011). Este se aplicará para el diseño de los componentes y propone tres capas fundamentales como lo indica su nombre, las cuales se describen a continuación según varios autores (Arevalo Lizardo, 2010) (Fernández Álvarez, 2008) (Galbán Izquierdo, 2007) .

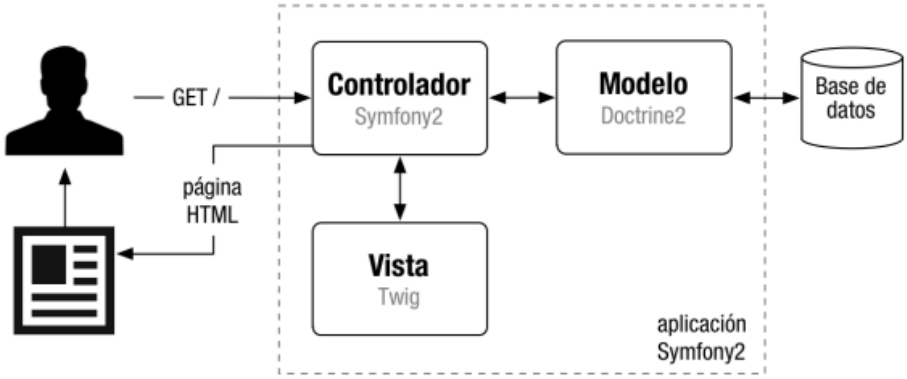

Figura 3: Uso del patrón Modelo-Vista-Controlador en Symfony 2.

Capa de presentación

La capa de presentación es la encargada a grandes rasgos de proveer la interacción con el usuario y facilitar funcionalidades como:

- Mostrar datos, formatearlos, ordenarlos.
- Solicitar datos, validarlos.
- Informar de los errores lógicos y de ejecución.
- Controlar la navegación entre pantallas.

Esta capa en el Módulo Trazas sería la encargada de visualizar al cliente las trazas registradas en la base de datos.

Capa de reglas de negocio

Esta capa contiene la funcionalidad que implementa la aplicación. Involucra cálculos basados en la información dada por el usuario y datos almacenados y validaciones. Controla la ejecución de la capa de acceso a datos y servicios externos. Se puede diseñar la lógica de la capa de negocios para uso directo por parte de componentes de presentación o su encapsulamiento como servicio y llamada a través de una interfaz de servicios que coordina la conversación con los clientes del servicio o invoca cualquier flujo o componente de negocio.

Esta capa en el Módulo Trazas sería la encargada de ajustar las funcionalidades para capturar los eventos y registrar las trazas. Se encontrarían también las clases controladoras encargadas de manejar la comunicación entre la capa de presentación y la capa de datos.

Capa de datos o persistencia

La capa de datos o persistencia no es más que un grupo de clases y de componentes responsables del almacenamiento de los datos, esta incluye necesariamente un modelo de las entidades del dominio del negocio. La capa de datos, es la representación real de los datos y representa además la persistencia del estado del sistema.

Esta capa representaría todas las clases entidades. Cubriría también aquellas clases o repositorios que almacenan las consultas para acceder a los registros de la base de datos.

2.6. Diagramas de clases del diseño

Los diagramas de clase constituyen una representación gráfica de las clases del sistema y sus relaciones. Visualizan un sistema desde diferentes perspectivas. Con el objetivo de obtener clases más independientes, reutilizables y fáciles de mantener se distribuyen en capas siguiendo el diseño que impone el estilo arquitectónico MVC. A continuación, se muestra el diagrama de clases correspondiente a los requisitos de listar trazas.

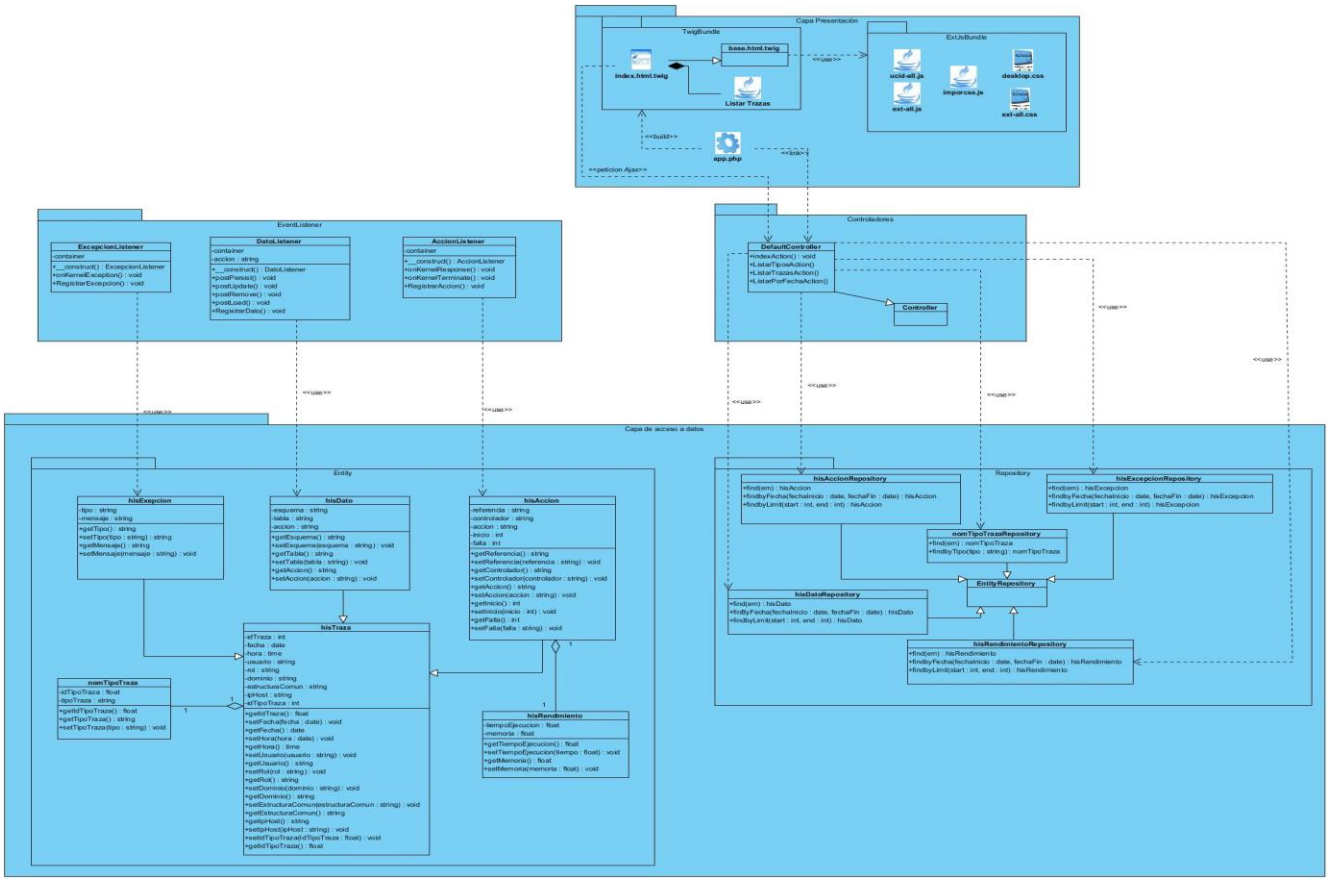

Figura 4: Diagrama de clases del diseño del módulo Trazas.

2.7. Diagramas de secuencia

Los diagramas de secuencia se utilizan para modelar los aspectos dinámicos de un sistema (Pressman, 2005). La mayoría de las veces, esto implica modelar instancias concretas o prototípicas de clases, interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento. A continuación se presenta el diagrama de secuencia del requisito Registrar Traza de Acción, los demás diagramas de secuencia se pueden consultar en los anexos al documento.

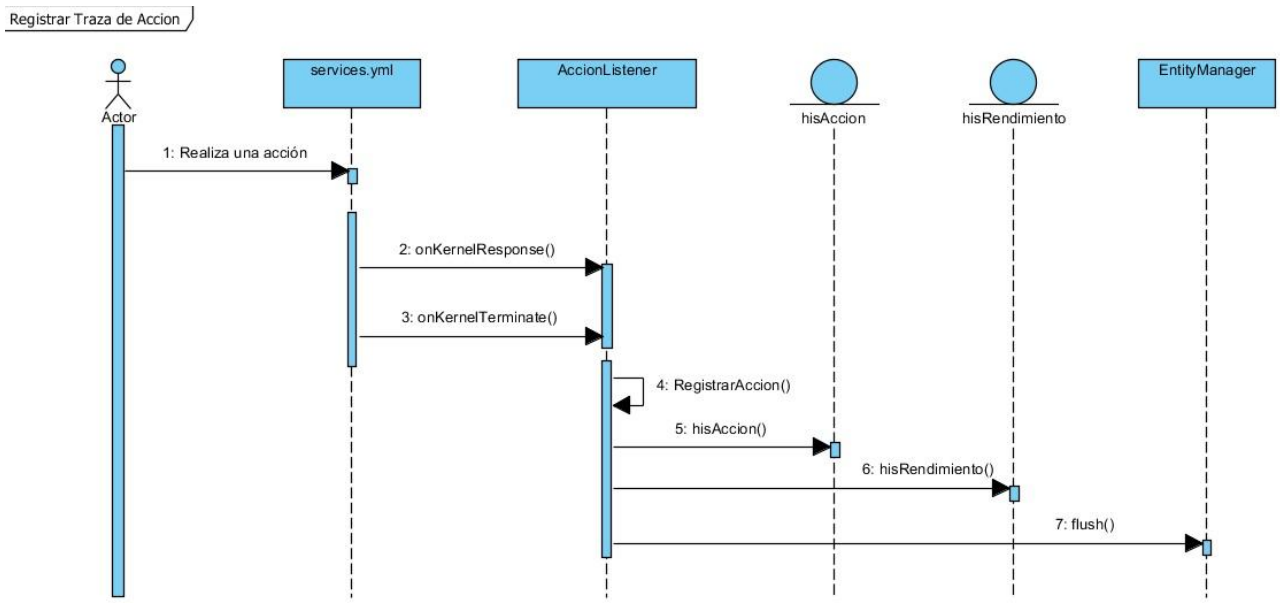

Figura 5: Diagrama de secuencia requisito Registrar Traza Acción

Modelado de la Base de Datos

El Diagrama de Entidad de Relación presentado en la Figura 6 se corresponde con la representación física de la base de datos del módulo Trazas. Expresa las entidades relevantes para el sistema así como sus interrelaciones y propiedades. El modelo de datos correspondiente al módulo de trazas tiene un total de 6 tablas, de las cuales 1 es nomenclador y las otras 5 constituyen las tablas que almacenan los datos de los diferentes tipos de trazas. Seguidamente se detallan las principales tablas de la base de datos, que se encuentran en el esquema mod_trazas.

Tabla 2: Descripción de las tablas de la base de datos del sistema.

Nombre Tabla	Descripción

his_traza	Almacena los datos que son comunes a todos los tipos de trazas.
his_accion	Almacena los datos correspondientes a la traza de acción.
his_dato	Almacena los datos correspondientes a la traza de datos.
his_excepcion	Almacena los datos correspondientes a la traza de excepción.
his_rendimiento	Almacena los datos correspondientes a la traza de rendimiento.
nom_tipotraza	Almacena los tipos de trazas que se gestionan.

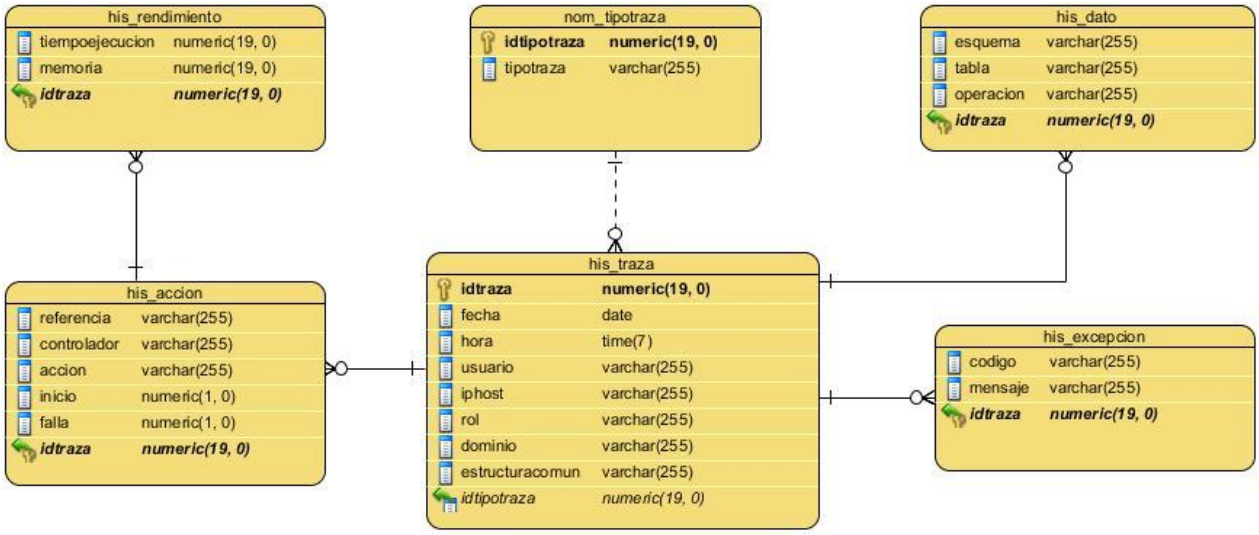

Figura 6: Diagrama de entidad de relación del módulo de Trazas.

2.8. Diagrama de Componentes

El diagrama de componentes muestra los elementos de diseño de un sistema de software, permite visualizar con más facilidad la estructura general del sistema y el comportamiento de los servicios que estos componentes proporcionan y utilizan a través de las interfaces. La propuesta de solución cuenta con una serie de componentes que interactúan entre sí. La ilustración siguiente muestra el diagrama de componentes elaborado.

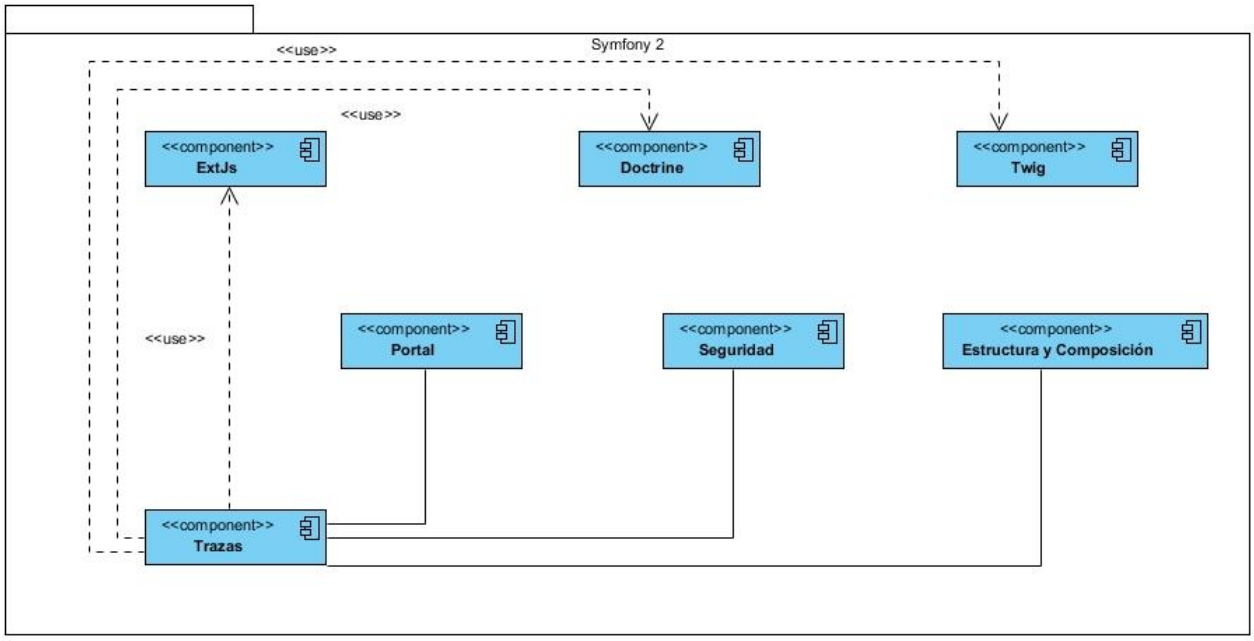

Figura 7: Diagrama de componentes

2.9. Diagrama de despliegue

El usuario desde una estación de trabajo podrá acceder al sistema utilizando un navegador web, el sistema estará desplegado en el mismo servidor web donde se encuentre ubicada la aplicación. Dicho servidor estará conectado a un servidor de bases de datos en el cual se almacenará la información de interés para la solución. A continuación se muestra el diagrama de despliegue:

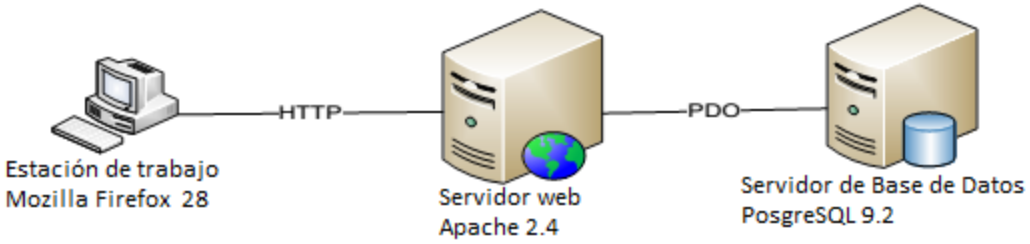

Figura 8: Diagrama de despliegue.

2.10. Conclusiones parciales

En este capítulo fueron mostrados los artefactos generados durante el diseño de la solución propuesta. En el mismo se identificaron y describieron los requisitos funcionales y no funcionales, se conformó la arquitectura del sistema que rige el diseño, el modelo de datos, el diagrama de clases del diseño propuesto, el diagrama de componentes y el de despliegue. El diseño e implementación del sistema se volvió complejo, puesto que no se contaba con la presencia de

especialistas que dominaran la materia del tema. Los requisitos fueron identificándose en el transcurso del desarrollo de la solución propuesta. Una vez concluido el diseño e implementación de la solución se prosigue a la implementación validación del sistema.

3. CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN.

Este capítulo tiene como objetivo evaluar y validar la calidad del sistema aplicando un conjunto de técnicas como las que se explicarán seguidamente. Se evaluará el diseño del sistema aplicando métricas de software para conocer cómo se afectan los valores de reutilización, acoplamiento, responsabilidad de las clases y complejidad de la implementación. También para evaluar la calidad del sistema se aplicarán pruebas de Aceptación con el objetivo de detectar y corregir el máximo de errores, antes de su entrega al cliente. Además se realizarán pruebas de caja blanca.

La IEEE plantea que la validación de software es: “el proceso de evaluar un sistema o componente, durante o al final del proceso de desarrollo, con el fin de determinar si satisface los requerimientos especificados”.

3.1. Métricas de software orientadas a clases para evaluar el diseño

Las métricas de software son una medida cuantitativa que permite a los desarrolladores tener una visión profunda de la eficacia del proceso del software y de los proyectos que dirigen utilizando el proceso como un marco de trabajo. Se reúnen los datos básicos de calidad y productividad. Estos datos son entonces analizados, comparados con promedios anteriores, y evaluados para determinar las mejoras en la calidad y productividad. Las métricas son también utilizadas para señalar áreas con problemas, de manera que se puedan desarrollar los remedios y mejorar el proceso del software (Pressman, 2005).

Debido a que este sistema se realizó bajo la programación orientada a objetos (POO) y las clases constituyen la unidad básica y fundamental en este tipo de programación, resulta viable realizar la validación del mismo con la aplicación de métricas dirigidas a sus clases de forma individual, sus jerarquías y colaboraciones.

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

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

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

Las métricas concebidas como herramienta para evaluar la calidad del diseño del módulo de Trazas y su relación con los atributos de calidad definidos en esta investigación son las siguientes.

3.1.1. Tamaño operacional de clases (TOC)

El tamaño operacional de las clases está dado por el número de métodos asignados una clase. Mediante el cual se calcula el nivel de Responsabilidad de los métodos, la Complejidad de implementación de los mismos y su Reutilización a fin de inspeccionar la efectividad del diseño, existiendo una relación directa con los dos primeros e inversa con el último antes mencionado.

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

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

Tabla 4: Criterios de evaluación para la métrica TOC.

Atributo	Categoría	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom y 2* Prom.
	Alta	> 2* Prom.
	Baja	< =Prom.

Complejidad de implementación	Media	Entre Prom y 2* Prom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Prom
	Alta	<= Prom.

3.1.2. Resultados obtenidos de la aplicación de la métrica TOC

El Gráfico 1 muestra la representación de los resultados obtenidos en la herramienta agrupados en los intervalos definidos. El gráfico refleja que la mayoría de las clases tienen de 1 a 5 procedimientos. Este resultado demuestra que el funcionamiento general del sistema está distribuido equitativamente entre la mayoría de las clases, de esta forma se garantiza que en caso de ocurrir algún cambio en el sistema de clases del componente, estaría menos comprometido el funcionamiento correcto del mismo.

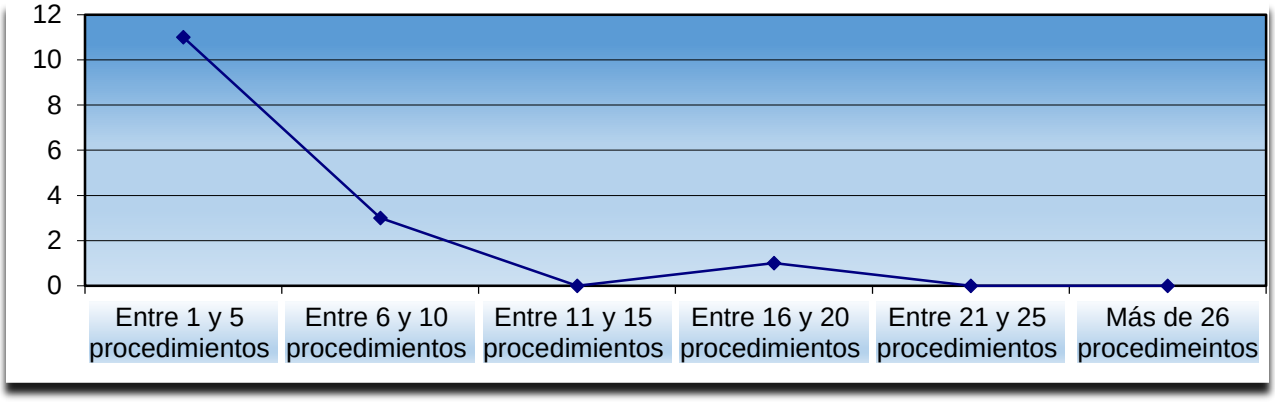

Figura 9: Gráfica correspondiente a la representación de la evaluación de la métrica TOC.

La Figura 10 muestra la representación en % de los resultados obtenidos en la herramienta agrupados en los intervalos definidos.

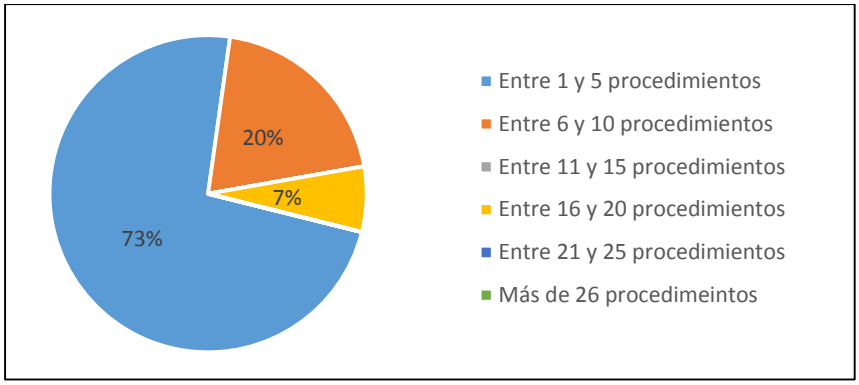

Figura 10: Gráfica correspondiente a la representación en % de la evaluación de la métrica TOC.

Estudiando los resultados de manera independiente se pueden realizar las siguientes valoraciones:

- Responsabilidad

La Figura 11 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad. Quedó demostrado que el 68% de las clases tienen una baja responsabilidad ya que este atributo se distribuyó equitativo entre todas las clases del sistema.

Tabla 5: Responsabilidad.

Responsabilidad	Cantidad de clases	Promedio
Baja	13	86,66666667
Media	1	6,666666667
Alta	1	6,666666667

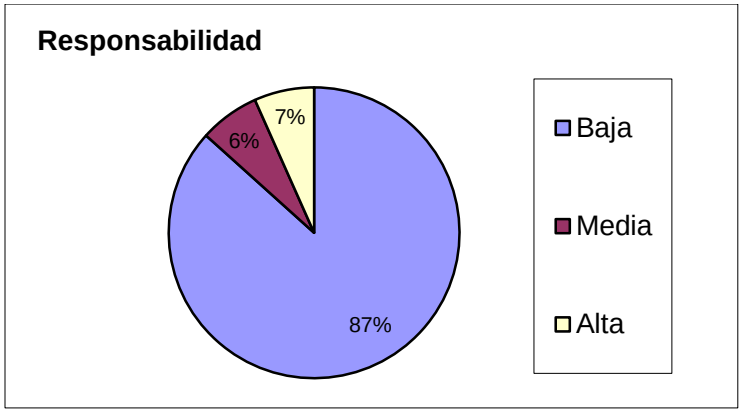

Figura 11: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

- Complejidad de implementación

La Figura 12 muestra la representación de la evaluación de la métrica TOC en el atributo Complejidad de implementación. El gráfico muestra que el 68% de las clases tienen una baja complejidad, este atributo está distribuido equitativamente. Esta característica permite mejorar el mantenimiento y soporte de estas clases.

Tabla 6: Complejidad de implementación.

Complejidad	Cantidad de clases	Promedio
Baja	13	86,66666667
Media	1	6,666666667
Alta	1	6,666666667

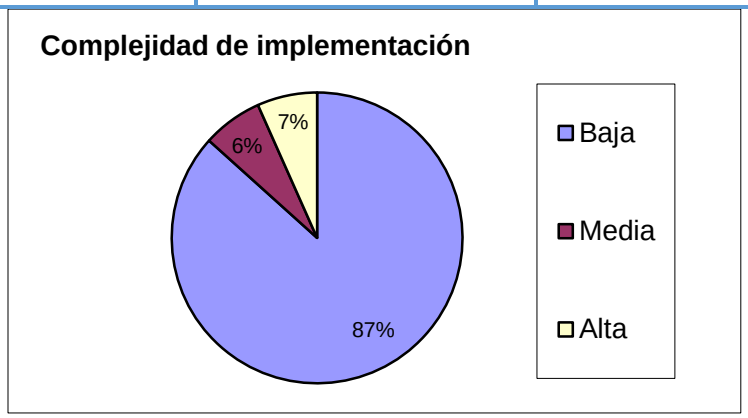

Figura 12: Gráfica que representa la evaluación de la métrica TOC en el atributo Complejidad de implementación.

- Reutilización

La Figura 13 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización. Quedando demostrado que el diseño de la solución es eficiente ya que la mayoría de las clases tienen un alto grado de reutilización.

Tabla 7: Reutilización

Reutilización	Cantidad de clases	Promedio
Baja	13	86,66666667
Media	1	6,666666667
Alta	1	6,666666667

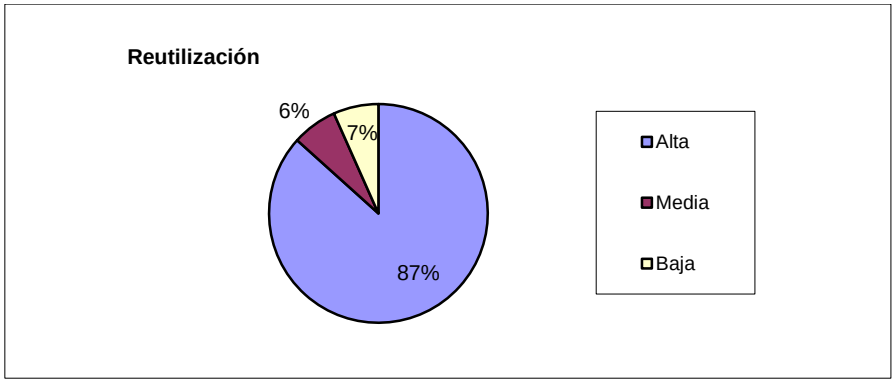

Figura 13: Gráfico que representa la evaluación de la métrica TOC en el atributo Reutilización.

Cuando existe un TOC alto se afectan los parámetros de calidad definidos por esta métrica. Se reduce la reutilización de las clases, la implementación se hace más compleja, las pruebas son difíciles de realizar y aumenta la responsabilidad de las clases.

Haciendo un análisis de los resultados obtenidos en la evaluación de la herramienta de medición de la métrica TOC, se puede concluir que el diseño del módulo Trazas tiene una calidad aceptable teniendo en cuenta que el 87 % de las clases incluidas en este sistema poseen una alta reutilización. Además este mismo por ciento de clases poseen evaluaciones positivas en los atributos de calidad (Responsabilidad y Complejidad de Implementación) en el diseño propuesto, evidenciando que las clases no tienen tanta responsabilidad, no son tan complejas, y poseen un elevado grado de reutilización. Por lo que se concluye que los resultados obtenidos según esta métrica son positivos.

3.1.3. Relaciones entre clases

Las relaciones entre las clases están dadas por el número de relaciones de uso que tenga una clase con otras, o sea el número de dependencias que una clase tiene con otra. Mediante la cual se calcula el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de pruebas a fin de inspeccionar la efectividad del diseño, existiendo una relación directa con los tres primeros e inversa con el último antes mencionado.

Tabla 8: Atributos de calidad evaluados por la métrica RC.

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

Tabla 9: Criterios de evaluación para la métrica RC.

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	> 2
Complejidad de mantenimiento	Baja	< =Prom.
	Media	Entre Prom y 2* Prom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom y 2* Prom
	Alta	<= Prom.
Cantidad de pruebas	Baja	< =Prom.
	Media	Entre Prom y 2* Prom.
	Alta	> 2* Prom.

3.1.4. Resultados obtenidos de la aplicación de la métrica RC

La Figura 14 muestra la representación de los resultados obtenidos en la herramienta agrupados en los intervalos definidos. Después de un análisis se decidió que no es posible eliminar estas dependencias por la responsabilidad arquitectónica que deben mantener las clases.

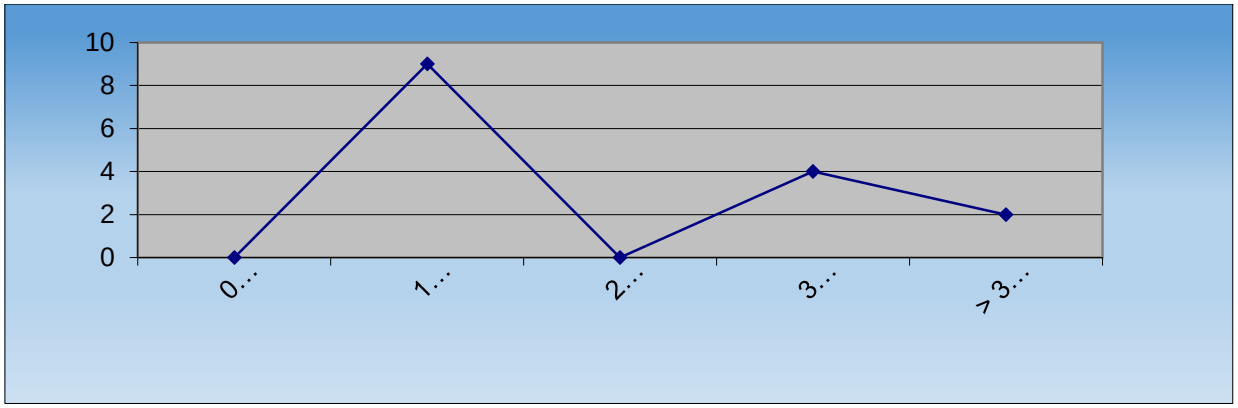

Figura 14: Representación de la evaluación de la métrica RC.

La Figura 15 muestra la representación en % de los resultados obtenidos en la herramienta agrupados en los intervalos definidos.

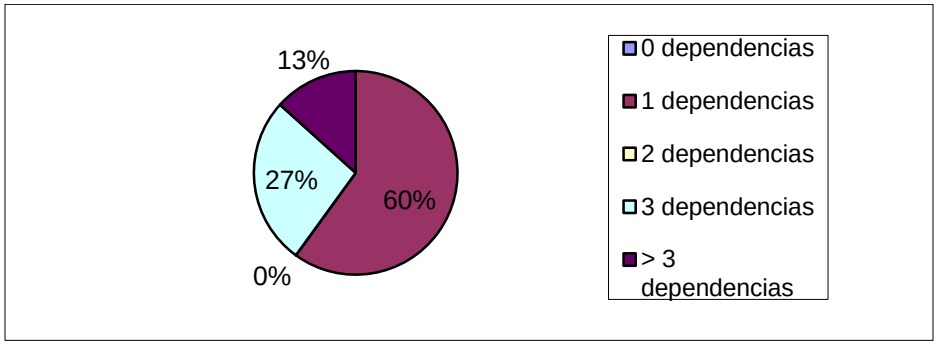

Figura 15: Representación en % de la evaluación de la métrica RC.

Analizando los resultados de forma independiente se pueden realizar las siguientes observaciones:

- Acoplamiento

El Figura 16 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento. Evidenciando un diseño eficiente al quedar reflejado que las clases cuentan con un bajo acoplamiento.

Tabla 10: Acoplamiento.

Acoplamiento	Cantidad de clases	Promedio
Ninguno	0	0
Bajo	9	60
Medio	0	0

Alto	6	40

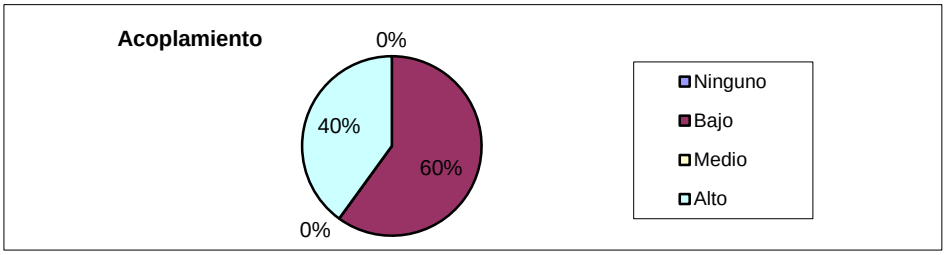

Figura 16: Representación de la evaluación de la métrica RC en el atributo Acoplamiento.

- Complejidad de mantenimiento

La Figura 17 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento. Mostrando la eficiencia de la arquitectura del sistema al evidenciarse una baja complejidad de mantenimiento o soporte.

Tabla 11: Complejidad de mantenimiento.

Complejidad de mantenimiento	**Cantidad de clases**	**Promedio**
Baja	9	60
Media	4	26,66666667
Alta	2	13,33333333

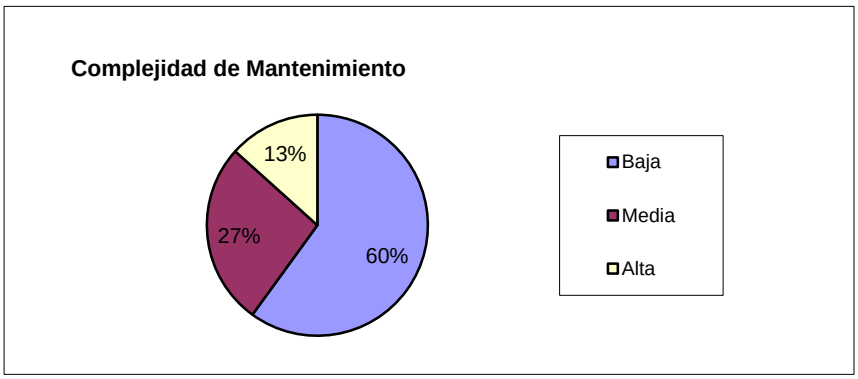

Figura 17: Representación de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento.

- Cantidad de pruebas

La Figura 18 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

Tabla 12: Cantidad de pruebas.

Complejidad de mantenimiento	Cantidad de clases	Promedio
Baja	9	60
Media	4	26,66666667
Alta	2	13,33333333

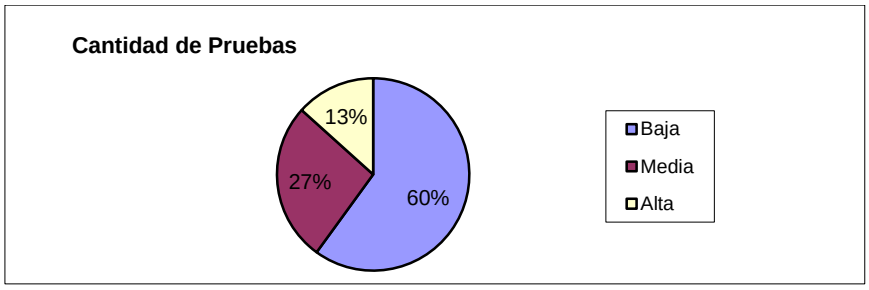

Figura 18: Representación de la evaluación de la métrica RC en el atributo Cantidad de pruebas.

- Reutilización

La Figura 19 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Tabla 13: Reutilización.

Reutilización	Cantidad de clases	Promedio
Baja	2	13,33333333
Media	4	26,66666667
Alta	9	60

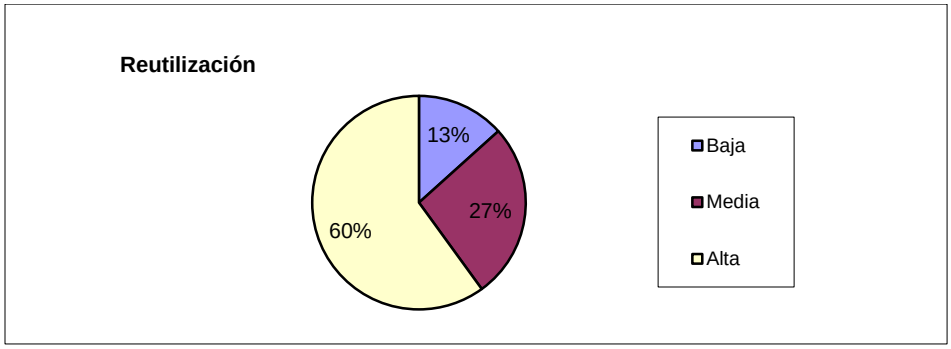

Figura 19: Representación de la evaluación de la métrica RC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación de la herramienta de medición de la métrica RC, se puede concluir que el diseño del módulo Trazas está entre los niveles de calidad requeridos, pudiéndose observar que el 87% de las clases posee entre 0 y 3 dependencias. Igualmente el 100% posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de mantenimiento, Reutilización y Cantidad de pruebas se comportan satisfactoriamente con un 60% de valor en las clases. Confirmando la elevada reutilización, el bajo acoplamiento, la baja complejidad y la baja cantidad de pruebas que se necesitan realizar.

3.2. Pruebas de software

Las pruebas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón, se debe definir en el proceso de la ingeniería del software una plantilla para las pruebas del software: un conjunto de pasos en los que podamos situar los métodos específicos de diseño de casos de prueba (Pressman, 2005).

Los casos de prueba especifican una forma de probar el sistema, incluyendo las entradas con las que se ha de probar, las condiciones bajo las que ha de probarse, así como los resultados esperados. Verifican el resultado de la interacción entre los actores y el sistema, validando así los requisitos funcionales que debe satisfacer el mismo. Este artefacto es generado durante el transcurso del desarrollo del software. El diseño de casos de prueba para la verificación del software puede significar un esfuerzo considerable. Los requisitos son la fuente principal para obtener los casos de prueba, para el desarrollo del módulo Trazas se obtuvo 1 caso de prueba respondiendo al requisito funcional Listar Trazas.

3.2.1. Pruebas de caja blanca

Las pruebas de caja blanca o pruebas estructurales son la forma de probar el correcto funcionamiento del código de un sistema informático. Seguidamente se describirá la realización de las pruebas, y las técnicas utilizadas: camino básico, cálculo de la complejidad ciclomática y extracción de los caminos independientes. Además se mostrarán los casos de pruebas elaborados y el análisis de los resultados obtenidos.

A continuación se muestra el código del método Listar Trazas al cual se le aplicó la métrica de complejidad ciclomática debido a que es uno de los métodos relevantes en la solución informática brindada:

```
public function ListarTrazasAction($tipo) {
    $limit = $_GET['limit'];
1   $start = $_GET['start'];
    $em = $this->container->get("doctrine")->getManager();

2   if ($tipo == 'Accion') {
3       $query = $em->getRepository('TrazasBundle:hisAccion')->findByLimit($em, $start, $limit);
        $cantReal = $em->getRepository('TrazasBundle:hisAccion')->find($em);
4   elseif ($tipo == 'Excepcion') {
5       $query = $em->getRepository('TrazasBundle:hisExcepcion')->find($em);
        $cantReal = $em->getRepository('TrazasBundle:hisExcepcion')->find($em);
6   elseif ($tipo == 'Datos') {
7       $query = $em->getRepository('TrazasBundle:hisDato')->find($em);
        $cantReal = $em->getRepository('TrazasBundle:hisDato')->find($em);
8   elseif ($tipo == 'Rendimiento') {
9       $query = $em->getRepository('TrazasBundle:hisRendimiento')->find($em);
        $cantReal = $em->getRepository('TrazasBundle:hisRendimiento')->find($em);
    }

10  $o = array(
        "num" => count($query),
        "data" => $query
    );
    return new Response(json_encode($o));
}
```

Figura 20: Código del método Listar Trazas

Grafo de flujo asociado:

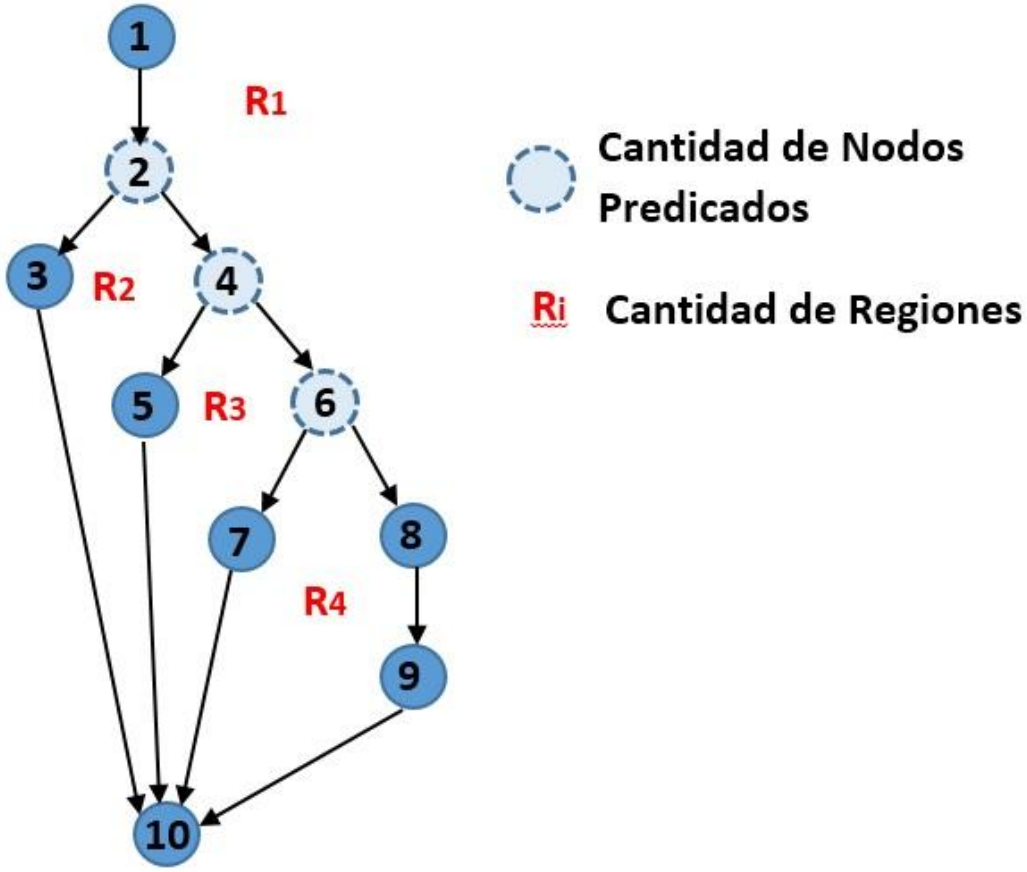

Figura 21: Grafo asociado al método Listar Trazas

Cálculo de complejidad ciclomática:

V (G) = R = 4

V (G) = A – N + 2 = 12 – 10 + 2 = 4

V (G) = P + 1 = 3 + 1 = 4

Caminos independientes determinados:

Camino 1: 1-2-3-10

Camino 2: 1-2-4-5-10

Camino 3: 1-2-4-6-7-10

Camino 4: 1-2-4-6-8-9-10

Para llevar a cabo las pruebas de caja blanca o pruebas estructurales se utilizó la herramienta PHPUnit. Además el código para la ejecución de las pruebas realizadas con dicha herramienta fue elaborado con el IDE Selenium. La herramienta seleccionada permite elegir sobre qué método específico o a qué clase se desea realizar una determinada prueba. Se realizaron las pruebas por cada uno de los caminos independientes determinados, obteniendo resultados satisfactorios.

3.2.2. Resultados de las pruebas de caja blanca

Para la validación del código generado en el desarrollo de la herramienta se seleccionaron los métodos más relevantes. A estos métodos se le realizaron pruebas para poder evaluar si el funcionamiento de cada uno de ellos se comportó de la manera esperada. Se realizaron pruebas a las 6 funcionalidades seleccionadas como relevantes, de las cuales 6 resultaron satisfactorias en una primera iteración de pruebas. Con la aplicación de las pruebas de caja blanca se obtuvo un 100% de pruebas con resultados satisfactorios, comprobándose la estabilidad de la lógica aplicada en el código.

3.2.3. Pruebas de caja negra

El enfoque de prueba aplicado fue el funcional o de Caja Negra el cual se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, nunca en la parte interna, es decir en el código de la solución. Este enfoque pretende demostrar, mediante los casos de prueba, que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como la integridad de la información externa se mantiene. Este tipo de prueba permite obtener un conjunto de condiciones de entrada que ejerciten de forma completa todos los requisitos funcionales de un programa, es un enfoque complementario que intenta descubrir diferentes tipos de errores como por ejemplo errores de interfaz, errores en estructuras de datos o acceso a las bases de datos entre otros.

Las pruebas de Caja Negra se centran en los requisitos funcionales del software, intentan encontrar errores de los siguientes tipos:

- Funciones incorrectas o inexistentes.
- Errores relativos a las interfaces.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores debidos al rendimiento.
- Errores de inicialización o terminación.

Para este enfoque de pruebas en este trabajo se empleó la técnica de Partición de Equivalencia la cual divide el campo de entrada de un programa en clases de datos. Una condición de entrada es un valor numérico específico, un rango de valores, un miembro de un conjunto de valores o lógica. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para una condición de entrada. El diseño de los casos de prueba para la partición de equivalencia se basa en la evaluación de las clases de equivalencia.

Dentro del método de Caja Negra la técnica de la Partición de Equivalencia es una de las más efectivas, pues permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La Partición de Equivalencia se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así en número de clases de prueba que hay que desarrollar.

3.2.4. Resultados de las pruebas de caja negra

Las pruebas de realizadas al sistema arrojaron un total de 6 no conformidades. Se muestra una tabla donde se refleja la cantidad de no conformidades detectadas por iteración. El sistema fue liberado en la 2da iteración.

Tabla 14: Resumen de No Conformidades por iteraciones.

Iteraciones	Cantidad de No Conformidades	Significativas	No significativas
Primera Iteración	2	2	0
Segunda Iteración	4	4	0
Total	6	6	0

3.2.5. Prueba de aceptación

El uso de cualquier producto de software tiene que estar justificado por las ventajas que ofrece. Sin embargo, antes de empezar a usarlo es muy difícil determinar si sus ventajas realmente justifican su uso. Uno de los mecanismos para esta determinación es la llamada prueba de aceptación. Las pruebas de aceptación se ejecutan antes de que la aplicación se instale en un entorno productivo, para brindarle al cliente la posibilidad de aceptar o rechazar el software desarrollado. Para ello se elaboran documentos de planes de prueba en los que se especifican

los criterios de aceptación a evaluar durante su ejecución (Lemus, 2012). La ejecución de dichas pruebas estará basada en un proceso denominado pruebas alfa y beta que a continuación se describe.

Pruebas alfa:

Son pruebas desarrolladas por el cliente en el lugar de desarrollo del sistema. Se usa el software con el desarrollador como observador del usuario y registrando los errores y problemas detectados.

Pruebas beta:

Se realizan luego de aplicar las pruebas alfa y se desarrolla en el entorno de trabajo del cliente. En este caso el cliente se queda a solas con el sistema y trata de encontrar fallos, de los cuales informa al desarrollador.

Para el desarrollo de las pruebas se elaboraron diferentes escenarios mediante los cuales el cliente puede interactuar con el sistema y probar las funcionalidades del mismo

3.2.6. Resultados de las pruebas de aceptación

Luego de aplicadas las pruebas el cliente emitió una carta de aceptación del software. Dicha carta se encuentra en los anexos del documento.

3.3. Conclusiones parciales

El presente capítulo se definieron algunos elementos que intervienen en el desarrollo de la solución propuesta, se aplicaron las métricas: Tamaño Operacional de la Clase y Relaciones entre Clases para evaluar la calidad el diseño propuesto. Los resultados arrojados por dichas métricas permitieron evaluar varios factores claves como la reutilización, complejidad de implementación, entre otros, de los cuales se obtuvo para las métricas TOC y RC una calidad aceptable.

También se efectuaron pruebas de caja negra al módulo Trazas para verificar su correcto funcionamiento. Tras 2 iteraciones de realización de pruebas funcionales se detectaron un total de 6 No Conformidades. Estas fueron resultas, permitiendo cumplir con los requisitos capturados en la primera etapa de desarrollo, luego de esto se aplicaron pruebas de aceptación, estas le permitieron al cliente emitir su valoración sobre el sistema desarrollado en función de si satisface sus necesidades o no.

Los resultados de las pruebas realizadas al sistema y la aplicación de las métricas en ambos casos descritos anteriormente dan lugar a que la implementación realizada presenta una calidad admisible, siendo satisfactoria la validación de la solución propuesta.

CONCLUSIONES GENERALES

A partir de un estudio realizado sobre los sistemas de gestión de trazas, se demostró que no existe una herramienta que se adapte completamente a las características del marco de trabajo Symfony 2 y satisfaga todas las necesidades del Departamento de Tecnologías del centro CEIGE, promoviendo así la búsqueda de una solución.

Se obtuvo un módulo para la gestión de las trazas desarrollado en Symfony 2, el cual brindará la posibilidad de mantener un registro de los eventos generados en los sistemas de gestión que se desarrollen en el centro CEIGE sobre el marco de trabajo Symfony 2. El mismo permitirá aumentar la seguridad en dichos sistemas garantizando que se pueda realizar procesos de auditorías sobre las aplicaciones. También se podrá hacer estudios en cuanto al rendimiento de las aplicaciones, determinando así cuales son las funcionalidades más críticas y trabajar en base a optimizarlas.

Se probó el módulo aplicando pruebas de Caja Blanca, Caja Negra y Aceptación, arrojando resultados satisfactorios. También se realizó una validación de la solución mediante la utilización de las métricas TOC y RC con el objetivo de comprobar la calidad del diseño propuesto, obteniéndose resultados positivos. Esto apoya el hecho de que la implementación desarrollada se puede considerar como aceptable, y el código puede ser reutilizado en futuras implementaciones. Por lo que se puede afirmar que se cumplió completamente el objetivo perseguido por la investigación.

RECOMENDACIONES

- Agregar al módulo Trazas las funcionalidades necesarias para registrar trazas de Integración, con el objetivo de tener un registro de los servicios web consumidos por la aplicación.
- Realizar un estudio detallado para determinar si la separación de los registros de las trazas, en una base de datos independiente, a la base de datos de la aplicación sobre la cual esté funcionando, optimiza el rendimiento de la misma. En caso de ser positivo el resultado, realizar las modificaciones pertinentes para lograrlo.

REFERENCIAS BIBLIOGRÁFICAS

(s.f.). Obtenido de UPM.es: http://is.ls.fi.upm.es/docencia/proyecto/docs/patrones_gof.pdf.

Achour, M., Betz, F., Dovgal, A., Nuno, L., Olson, P., Richter, G., & Seguy, D. A. (2006). *PHP Manual.*

AIRESProxy. (2014). Obtenido de https://srni2.uci.cu/bienvenida

Alegsa.com.ar. (s.f.). Obtenido de http://www.alegsa.com.ar/Dic/framework.php

Arevalo Lizardo, M. E. (2010). *Introducción al Patrón de Arquitectura por Capas.*

Aruquipa Chambi, M. G. (2007). *Desarrollo de software basado en componentes.* Universidad Mayor de San Andrés, La Paz, Bolivia.

Baryolo Gómez, O. y. (2012). *Modelo de gestión de log para la auditoría de información de apoyo a la toma de decisions en las organizaciones.* [En línea] http://scielo.sld.cu.

Bolton, D. (2014). *Definition of IDE.*

Cañete Pupo, Y. (2010). *Libro de Ayuda del Marco de trabjo Sauxe, en su versión 2.0.* Ciudad de la Habana.

CornerBowl Software. (2014). Obtenido de http://www.cornerbowl.com/Log-Manager/Log-Manager.aspx

Corrales Martínez, Y. (2009). *Componentes para la configuración de la gestión del multilenguaje y la gestión de trazas del sistema Cedrux.* Ciudad de la Habana.

Date, C. J. (2001). *Introducción a los SISTEMAS DE BASES DE DATOS: Séptima Edición.*

Doctrine Project Team. (03 de 11 de 2011). Obtenido de http://www.lawebdelprogramador.com/cursos/PHP/7033-Doctrine_2_ORM_Documentation.html

Drake, J. M. (2008). *Proceso de desarrollo de aplicaciones de software.* Obtenido de CTR-Computadores y Tiempo Real: http://www.ctr.unican.es/asignaturas/MC_OO/Doc/OO_08_I2_Proceso.pdf

Eguiluz, J. (2011). *Desarrollo web ágil con Symfony 2-Primera edición.*

Facultad de Ciencias, UNAM. (2002). *Universidad Nacional Autónoma de México.* Obtenido de http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html

Fernández Álvarez, A. M. (2008). *Arquitectura de aplicaciones.*

Fernández Díaz, M. (15 de 12 de 2012). *Serie Científica de la Universidad de Ciencias Informáticas.* Obtenido de http://publicaciones.uci.cu/index.php/SC/article/viewFile/1049/603

Firefoxmania. (2014). Obtenido de http://firefoxmania.uci.cu/download/?p=faq&a=fx

Frederick, S. R. (2010). *Learning Ext JS.*

Galbán Izquierdo, Y. (2007). *Arquitectura de los Módulos Entrada de Datos y Generador de Modelos, del Sistema Integrado de Gestión Estadística. Universidad de las Ciencias Informáticas.*

Gamma, E. y. (s.f.). *Design Patterns: Elements of Reusable Object -Oriented Software.*

González Obregón, W. (2012). *CEIGE: Subdirección de Producción: Modelo de Desarrollo de Software.* Ciudad de la Habana.

IEEE. (2004). *Guía al cuerpo de conocimiento de la Ingeniería de Software.*

Larman, C. (1999). *UML y Patrones, Una introducción al análisis y diseño orientado a objetos y al proceso unificado. 2da Edición.*

LIBROSWEB.es. (2010). Obtenido de http://librosweb.es/symfony/capitulo_8/por_que_utilizar_un_orm_y_una_capa_de_abstraccion.html

Mark. (2010). *Learning Python, Fourth Edition.*

Mendez, G. (2009). *Facultad de Informática. Universidad Complutense de Madrid.* Obtenido de Proceso de Software y Ciclo de Vida: https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/02-ProcesoCicloDeVida.pdf

Microsoft Developer Network. (2014). Obtenido de http://msdn.microsoft.com/es-es/library/ms172144%28v=vs.110%29.aspx

Morales, Y. R., Marrero, M. E., & Oliva, A. T. (2013). Extensión de la herramienta Visual Paradigm para la generación de clases de acceso a datos con Doctrine 2.0. *Serie Científica de la Universidad de las Ciencias Informáticas*.

Mozilla. (2014). Obtenido de http://www.mozilla.org/es-ES/firefox/features/

Mozilla Developer Network. (2011). Obtenido de https://developer.mozilla.org/es/docs/JavaScript/Acerca_de_JavaScript

NetBeans. (2014). Obtenido de https://netbeans.org/index_es.html

Pacheco, N. (2011). *Manual de Twig.*

Pérez Sarduy, M. M. (2009). *Propuesta de modelo de desarrollo de software tecnológico del Centro de Soluciones de Gestión.* Ciudad de la Habana.

pgAdmin. (2014). Obtenido de http://www.pgadmin.org/

PostgreSQL-es. (2014). Obtenido de http://www.postgresql.org.es/sobre_postgresql

Pressman, R. S. (2005). *Ingeniería de Software. Un enfoque práctico.* New York: Mac Graw Hill. Higher Education.

Quintero, J. B. (2010). Arquitectura de Software. Patrones en la arquitectura.

Ramirez, A. (2004). *Introducción a los patrones de diseño.*

Reynoso, C. A. (2004). *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*

Rincón Carrera, J. M. (2011). *Estudio de la Evolución Web y Lenguajes Dinámicos.* Universidad Carlos III de Madrid.

Sacerio Martínez, A. (2010). *Análisis, diseño e implementación del Framework de registro de eventos y su herramienta de seguimiento.* Ciudad dela Habana.

Sánchez Asenjo, J. (2013). *Gestión de base de datos .*

Schmuller, J. P. (2001). *Aprediendo UML en 24 horas.* Obtenido de http://www.intercambiosvirtuales.org/libros-manuales/aprendiendo-uml-en-24-horas-joseph

SEI, Software Engennering Insttitute. (2014). Obtenido de http://www.sei.cmu.edu

Sencha. (2014). Obtenido de http://www.sencha.com/products/extjs/

Serradilla, J. L. (2007). *Universidad de Murcia.* Obtenido de http://www.um.es/atica/documentos/PREsubversion.pdf

Sommerville, I. (2005). *Ingeniería de Software.* Madrid: Pearson Educacion S.A.

Symfony.es. (2010). Obtenido de http://symfony.es/noticias/2010/09/01/symfony2-presenta-su-webprofiler/

symfony.es. (2014). Obtenido de http://symfony.es/que-es-symfony

Tomar, N. (2011). *Computer Aided Software Engineering Tools (CASE).*

TortoiseSVN. (2014). Obtenido de http://tortoisesvn.net/docs/nightly/TortoiseSVN_es/tsvn-preface-features.html

Uci. (2014). Obtenido de http://www.uci.cu/?q=mision

Wampserver. (2014). Obtenido de http://www.wampserver.com/

Anexo 1: Descripción del requisito Registrar Traza de Datos.

Tabla 15: Descripción del requisito Registrar Traza de datos

Precondiciones	Un usuario inicia sesión en el sistema.
Flujo de eventos	
Flujo básico Registrar traza de datos	
1	El usuario realiza alguna acción que desencadena alguna operación sobre la base de datos, ya sea de inserción, modificación o eliminación.
2	Se capturan los datos de la traza: -Fecha -Dominio -Hora -Estructura común -Usuario -Esquema -Ip host -Tabla -Rol -Acción
3	Concluye el requisito
Post-condiciones	
1	Se persisten los datos en la base de datos
Flujos alternativos	
Flujo alternativo <<Nº Evento>>.<<letra iniciando por la a>> <Condición que dio lugar a la extensión>	
1	
Post-condiciones	
1	
Validaciones	
1	Se validan los datos según lo establecido en el Modelo conceptual CEIGE_Modelo_conceptual_Trazas.

Conceptos	**Traza de datos**	Utilizados internamente: -Fecha -Dominio -Hora -Estructura común -Usuario -Esquema -Ip host -Tabla -Rol -Acción
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Anexo 2: Descripción del requisito Registrar Tazas de Excepción

Tabla 16: Descripción del requisito Registrar Traza de Excepción

Precondiciones	Un usuario inicia sesión en el sistema	
Flujo de eventos		
Flujo básico Registrar traza de excepción		
1	Se produce un error en la ejecución de una acción.	
2	Se capturan los datos de la traza: -Fecha -Estructura común -Hora -Código -Usuario -Tipo -Ip host -Mensaje -Rol -Descripción -Dominio -Log	
3	Concluye el requisito	
Post-condiciones		
1	Se persisten los datos en la base de datos.	
Flujos alternativos		
Flujo alternativo <<Nº Evento>>.<<letra iniciando por la a>> <Condición que dio lugar a la extensión>		

1			
Post-condiciones			
1			
Validaciones			
2	Se validan los datos según lo establecido en el Modelo conceptual CEIGE_Modelo_conceptual_Trazas.		
Conceptos	**Traza de Excepción**	Utilizados internamente: -Fecha -Hora -Usuario -Ip host -Rol	-Dominio -Estructura común -Tipo -Mensaje
Requisitos especiales	N/A		
Asuntos pendientes	N/A		

Anexo 3: Descripción del requisito Registrar Tazas de Rendimiento.

Precondiciones	Un usuario inicia sesión en el sistema
Flujo de eventos	
Flujo básico Registrar traza de rendimiento	
1	El usuario realiza alguna acción en el sistema.
2	Se capturan los datos de la traza: -Tiempo de ejecución -Memoria
3	Concluye el requisito
Post-condiciones	
2	Se persisten los datos en la base de datos.
Flujos alternativos	
Flujo alternativo <<Nº Evento>>.<<letra iniciando por la a>> <Condición que dio lugar a la extensión>	

1		
Post-condiciones		
1		
Validaciones		
1	Se validan los datos según lo establecido en el Modelo conceptual CEIGE_Modelo_conceptual_Trazas.	
Conceptos	**Traza de rendimiento**	Utilizados internamente: -Tiempo de ejecución -Memoria
Requisitos especiales	N/A	
Asuntos pendientes	N/A	

Anexo 4: Diagrama de secuencia del requisito Registrar Traza de Datos.

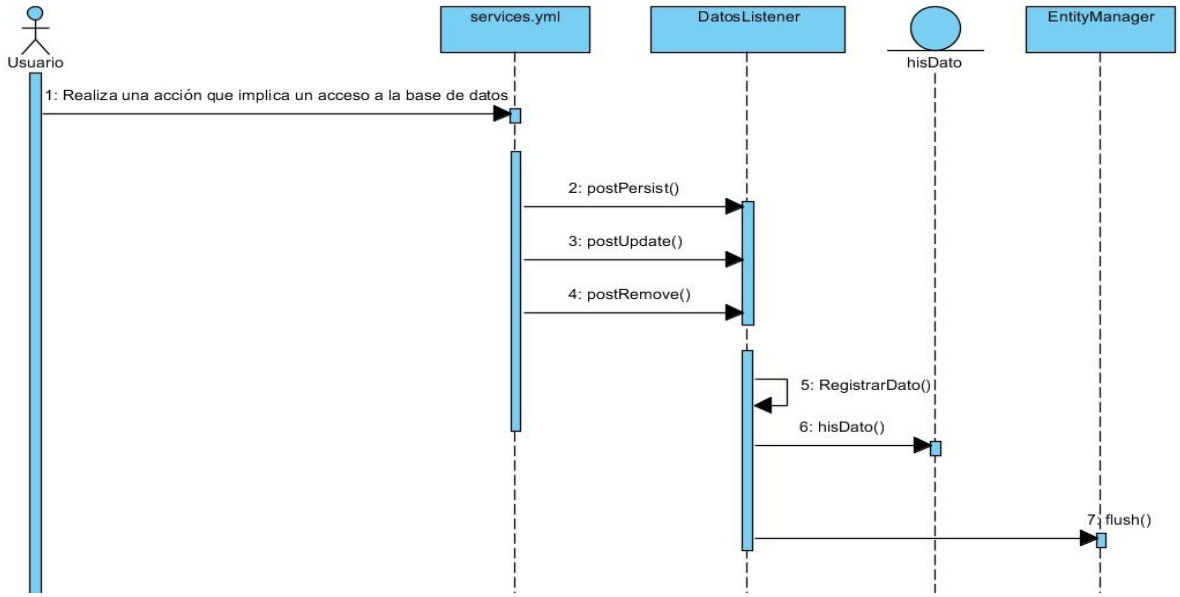

Figura 22: Diagrama de secuencia del requisito Registrar Traza de Datos

Anexo 5: Diagrama de secuencia del requisito Registrar Traza de Excepción.

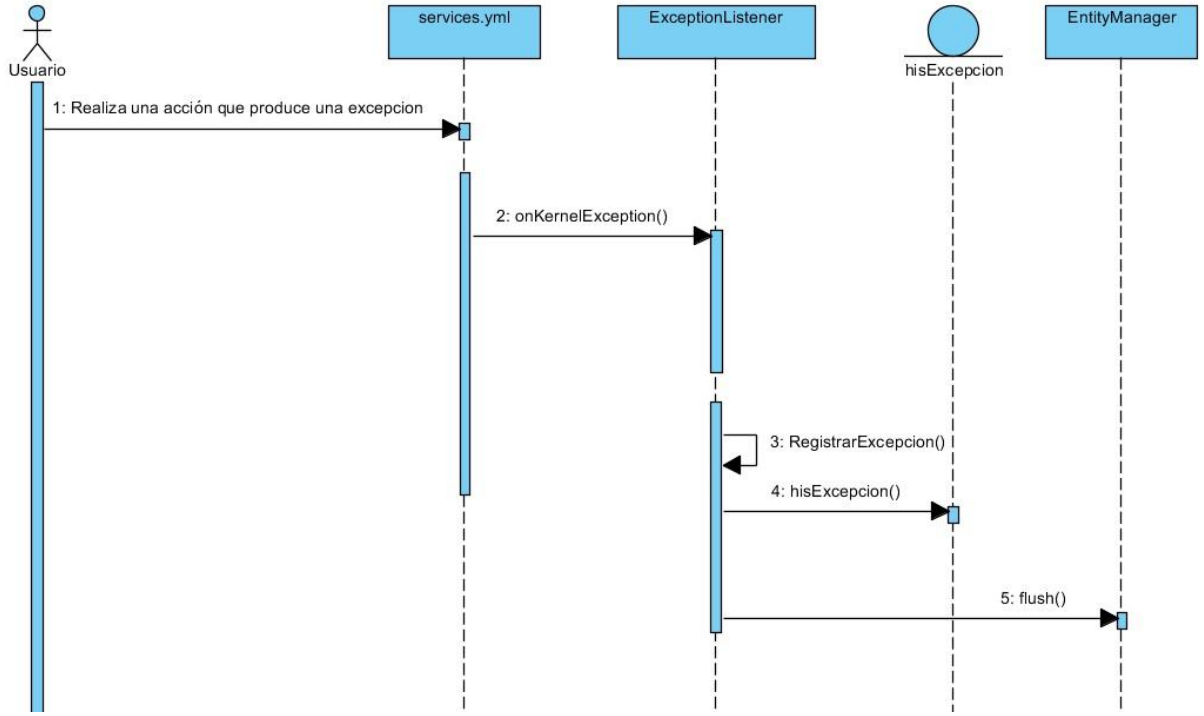

Figura 23: Diagrama de Secuencia del requisito Registrar Traza de Excepción