

# Universidad de las Ciencias Informáticas

## Facultad 3



**Título:** *Componente “Gestor de Trazas” para el Sistema de Informatización de la Gestión de las Fiscalías.*

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

**Autores:**

- Lisandra Leyva Leal.
- Frank Lemus Paz.

**Tutores:**

- Yenier Figueroa Machado.
- Eduardo Castillo Benítez.

La Habana, Cuba

2012



## **DECLARACIÓN DE AUTORÍA**

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

---

Lisandra Leyva Leal

Autor

---

Frank Lemus Paz

Autor

---

Ing. Yenier Figueroa Machado

Tutor

---

Ing. Eduardo Castillo Benítez

Co - Tutor



**Tutor:** Ing. Yenier Figueroa Machado.

- ✓ Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI).
- ✓ Profesor de la Universidad de las Ciencias Informáticas, en la Disciplina de Programación.
- ✓ Jefe del Proyecto Productivo Sistema de Informatización de la Gestión de las Fiscalías de la Facultad 3.
- ✓ Correo electrónico: [yfigueroa@uci.cu](mailto:yfigueroa@uci.cu).

**Co-Tutor:** Ing. Eduardo Castillo Benítez

- ✓ Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI).
- ✓ Arquitecto principal del proyecto Sistema de Informatización de la Gestión de las Fiscalías de la Facultad 3.
- ✓ Correo electrónico: [ecbenitez@uci.cu](mailto:ecbenitez@uci.cu).



**Dedicatoria**

*A mis padres por permitirme estar aquí, por todo su amor, abnegación, ejemplo y sacrificio. En especial a mi mamá luz y motor impulsor de mi vida. Todos mis logros son gracias y para ella.*

*A mi padre Adalberto, soporte, confianza y ejemplo en mi vida, para ti que desde el cielo estás orgulloso de mi triunfo.*

*A mi abuela Alicia, ausente físicamente pero presente siempre en mi corazón.*

*Lisandra*

*Quiero dedicar este trabajo a mis padres Elena y Frank, que me han sabido guiar desde pequeño y gracias a ellos he logrado muchos de mis sueños que nunca pensé alcanzar. A mi familia, en especial a mi abuela a la que tanto quiero.*

*A mis mejores amigos que me han ayudado en los momentos más difíciles en estos 5 años, y me han apoyado incondicionalmente, gracias por estar ahí.*

*Frank*



*Especialmente a mis padres por darme siempre todo su amor y apoyo, en las buenas y malas decisiones, por estar ahí en los momentos felices y tristes, por creer en mí y por la educación que me han dado para ser siempre una mejor persona.*

*A mi novio Rances por su amor, respeto y fuerza, por estar ahí siempre cuando más lo necesité. A mi otra madre Aly por su ayuda y soporte todos estos años, por creer en mí.*

*A toda mi familia, mi tío Ramón y Toñito ejemplo de superación y consagración, mi abuelo Antonio al cual le parecieron una eternidad estos cinco años lejos de casa, a mi madrecita Saily por toda la fuerza y ayuda toda mi vida. A Arnaldo, Cruz, Carmen y toda la familia Fuentes por educarme también en los primeros años de vida y ser parte de mi familia también. A Belkis y Elider por su apoyo y cariño siempre.*

*A mi hermana Aimé por ser siempre mi fuerza, confidente y ejemplo en todos los momentos de mi vida. A mis amistades Yaimí, Cheng, Yuniel, Yasmany y todos los demás, por compartir conmigo todos los tiempos, clases, fiestas y tristezas.*

*A amigos incondicionales que me han apoyado estos cinco años de Universidad desde sus inicios como Anaelys, gracias por estar ahí, ser amiga y apoyo siempre que lo necesité. Ariadna, Yoandra, Julio, Driggs, Milena, Javier, Yasmany, el Yoss y Daniel por compartir todos estos años desde el primero. A mis amigos del nueve y el diez.*

*A Doa, Irmel, el Yosle, por soportarme todos estos años y ser parte de mi familia, por los tiempos de fiestas y tragos amargos. A Deisy y Lisbeth amigas de sacrificios, buenos y malos tiempos. A mis tutores por el apoyo en este trabajo.*

*A todos ellos GRACIAS*

*Lisandra*



*Son muchas las personas que me han ayudado, tanto en la realización de este trabajo como en mi tránsito por esta maravillosa universidad. A todos ellos: GRACIAS.*

*A mis padres, ejemplos de abnegación y cariño de los cuales me siento profundamente orgulloso de ser hijo, y que sin la ayuda de ellos nada hubiese sido posible y mi familia que mejor no podría pedir.*

*A todos mis profesores que en todos estos años me han impartido sus conocimientos con la mejor disposición y con los cuales he aprendido y me he formado.*

*A mis amigos y compañeros a lo largo de estos 5 años, con los cuales he reído, compartido, convivido como si fuésemos hermanos y muchos de ellos me han dado su apoyo y me han ayudado en muchas ocasiones. En especial a mi amigo Felipe que me ayudó mucho en la realización de este trabajo, entre otros compañeros.*

*A los tutores Figueroa, Eduardo y a todas aquellas personas que de una manera u otra han influenciado en mi formación durante estos largos años de estudio, mis más profundo e infinito agradecimiento.*

*Frank*



## Resumen

En la actualidad una característica que deben tener presente las aplicaciones informáticas que se dedican a la gestión de datos e información es la capacidad de ser auditables. La necesidad de controlar la información que se maneja, así como llevar un registro detallado de la misma, posibilita una correcta gestión del uso del sistema. Haciendo uso de las trazas las cuales llevan un registro de las acciones que va dejando un usuario o sistema a lo largo de la aplicación podemos lograr este fin.

En el proyecto Sistema de Informatización de la Gestión de las Fiscalías (SIGEF II) desarrollado por estudiantes y profesores de la Universidad de las Ciencias Informáticas (UCI) con el objetivo de gestionar los procesos de la Fiscalía General de la República de Cuba; se maneja información clasificada y restringida, y no se cuenta con una herramienta que permita monitorear y registrar los pasos de un usuario, las acciones que realiza y los posibles errores o excepciones que puedan ocurrir debido a alguna falla del sistema.

El presente trabajo recoge información sobre la realización de un componente para el control y registro de las trazas, facilitando la evaluación de forma independiente y objetiva de todos los aspectos del entorno informático para el cual fue desarrollado, contribuyendo así a comprobar si el mismo está cumpliendo con los objetivos de la organización.

Para satisfacer la necesidad que posee el proyecto SIGEF II de ser auditable se realiza un estudio de herramientas y mecanismos de gestión de trazas recogiendo los principales requisitos que el componente propuesto debe tener. Luego de detallar cada uno de los requisitos se realizó el diseño del componente, así como la evaluación de cada uno de los resultados obtenidos.

Con la implementación de esta propuesta se espera que los fiscales dispongan de una herramienta que les facilite el control y manejo de la información referente a los procesos informatizados en el sistema.

### PALABRAS CLAVE

Trazas, Symfony, auditoría, control y registro de las trazas.



ÍNDICE DE CONTENIDOS

<b>Agradecimientos</b> .....	<b>4</b>
<b>Dedicatoria</b> .....	¡Error! Marcador no definido.
<b>Resumen</b> .....	<b>7</b>
<b>Introducción</b> .....	<b>14</b>
<b>Capítulo 1: Fundamentación teórica</b> .....	<b>18</b>
<b>1.1 Conceptos Fundamentales</b> .....	<b>18</b>
<b>1.2 Características del proyecto Sistema de Informatización de la Gestión de las Fiscalías (SIGEF II)</b> . .....	<b>19</b>
<b>1.3 Manejo de las trazas a nivel internacional y en la industria cubana del software.</b> .....	<b>22</b>
<b>1.4 Principales herramientas de registro de trazas.</b> .....	<b>23</b>
1.4.1 <i>TraceListener</i> .....	23
1.4.2 <i>Audit Logger 5.5</i> .....	24
1.4.3 <i>CRM Logger</i> .....	25
1.4.4 <i>Acaxia para el marco de trabajo SAUXE</i> .....	25
<b>1.5 Mecanismo de control de las trazas de Symfony</b> .....	<b>26</b>
<b>1.6 Propuesta de solución</b> .....	<b>28</b>
1.6.1 <i>Metodología de desarrollo de software (6)</i> .....	29
1.6.2 <i>Arquitectura del componente</i> .....	31
1.6.3 <i>Herramientas y tecnologías definidas para la realización del componente:</i> .....	32
1.6.3.1 <i>Visual Paradigm</i> .....	33
1.6.3.2 <i>Ajax con JQuery</i> .....	33
1.6.3.3 <i>Propel</i> .....	34
1.6.3.4 <i>Gestor de Base de Datos</i> .....	34
1.6.3.5 <i>Prototipado</i> .....	35





1.6.3.6 Lenguaje de Programación.....	35
1.6.3.7 IDE de desarrollo .....	36
1.6.3.8 Marco de trabajo.....	36
<b>1.7 Métricas.....</b>	<b>37</b>
1.7.1 Métricas para el diseño .....	37
1.7.1.1 Tamaño Operacional de Clase (TOC):.....	38
1.7.1.2 Relaciones entre clases (RC):.....	38
<b>1.8 Conclusiones Parciales.....</b>	<b>39</b>
<b>Capítulo 2: Características del Sistema. ....</b>	<b>40</b>
<b>2.1 Especificación de los requerimientos del software.....</b>	<b>40</b>
2.1.1 Requerimientos no funcionales .....	40
2.1.2 Requerimientos funcionales: .....	42
<b>2.2 Etapa de Planificación.....</b>	<b>42</b>
2.2.1 Historias de Usuario .....	43
2.2.2 Plan de iteración.....	44
2.2.3 Plan de entregas.....	47
<b>2.3 Etapa de Diseño .....</b>	<b>48</b>
2.3.1 Patrones de Diseño.....	48
2.3.2 Tarjetas CRC (Cargo o clase, Responsabilidad y Colaboración).....	50
2.3.3 Diseño de la Base de Datos. ....	51
2.3.4 Diagrama de Despliegue .....	52
<b>2.4 Etapa de Desarrollo .....</b>	<b>53</b>
2.4.1 Estándares de codificación .....	53
2.4.2 Programación en parejas .....	54
2.4.3 Tareas de la ingeniería .....	54



2.4.4 Pruebas.....	55
2.4.4.1 Pruebas Unitarias .....	56
2.4.4.2 Pruebas de aceptación .....	56
<b>2.5 Conclusiones Parciales.....</b>	<b>56</b>
<b>Capítulo 3: Pruebas del Sistema. ....</b>	<b>58</b>
<b>3.1 Métricas de Software .....</b>	<b>58</b>
3.1.1 TOC ( <i>Tamaño Operacional de Clase</i> ).....	58
3.1.2 RC ( <i>Relaciones entre Clases</i> ).....	60
<b>3.2 Pruebas de Aceptación .....</b>	<b>63</b>
<b>3.3 Pruebas Unitarias .....</b>	<b>64</b>
3.3.1 Camino Básico .....	64
3.3.2 Caso del método <i>getObtenerMódulo</i> :.....	65
3.3.2.1. Grafo de Flujo:.....	66
3.3.2.2 Complejidad Ciclomática:.....	66
3.3.2.3 Interfaz de prueba.....	67
3.3.3 Caso del método <i>executePrincipal</i> : .....	69
3.3.3.1 Grafo de Flujo:.....	70
3.3.3.2 Complejidad Ciclomática:.....	70
3.3.4 Análisis de resultados.....	71
<b>3.4 Conclusiones Parciales.....</b>	<b>71</b>
<b>Conclusiones Generales .....</b>	<b>72</b>
<b>Recomendaciones .....</b>	<b>73</b>
<b>Bibliografía.....</b>	<b>74</b>



ÍNDICE DE FIGURAS

Figura 1: Estructura del proyecto Sistema de Informatización de la Gestión de las Fiscalías. .... 20

Figura 2: Fragmento del contenido de un log de Symfony 1.3 ..... 27

Figura 3: Ciclo completo de XP..... 31

Figura 4: Descripción del plan iteración #1..... 47

Figura 5: Ejemplo de uso del patrón Decorator en el componente..... 48

Figura 6: Ejemplo de uso del patrón Observer en el componente. .... 49

Figura 7: Ejemplo de uso del patrón Creator en el componente..... 50

Figura 8: Diagrama Entidad – Relación. .... 52

Figura 9: Diagrama de despliegue..... 53

Figura 10: Resultados de la evaluación de la métrica TOC para el atributo Responsabilidad. .... 59

Figura 11: Resultados de la evaluación de la métrica TOC para el atributo Complejidad. .... 60

Figura 12: Resultados de la evaluación de la métrica TOC para el atributo Reutilización..... 60

Figura 13: Resultados de la evaluación de la métrica RC para el atributo Acoplamiento. .... 61

Figura 14: Resultados de la evaluación de la métrica RC para el atributo Complejidad de Mantenimiento. .... 62

Figura 15: Resultados de la evaluación de la métrica RC para el atributo Cantidad de Pruebas. .... 62

Figura 16: Resultados de la evaluación de la métrica RC para el atributo Reutilización. .... 62

Figura 17: Método getObtenerModulo. .... 65

Figura 18: Grafo de Flujo del método getObtenerModulo..... 66

Figura 19: Grafo de Flujo del método getObtenerModulo dividido por regiones..... 67

Figura 20: Caso de prueba del método getObtenerModulos. .... 68

Figura 21: Caso de prueba del método getObtenerModulos. .... 68

Figura 22: Caso de prueba del método getObtenerModulos. .... 69

Figura 23: Método executePrincipal..... 69



**Índice de figuras**

Figura 24: Grafo de Flujo del método executePrincipal.....	70
Figura 25: Grafo de Flujo del método executePrincipal dividido por regiones. ....	70



---

**Índice de tablas**

Tabla 1: Análisis comparativos de las herramientas de trazas. ....	28
Tabla 2: Representación de la Historia de Usuario #3. ....	44
Tabla 3: Representación de la Historia de Usuario #4. ....	44
Tabla 4: Distribución de las tareas por cada Historia de Usuarios.....	45
Tabla 5: Plan de entregas de las iteraciones. ....	47
Tabla 6: Descripción de la CRC Traza.....	51
Tabla 7: Descripción de la CRC Mi Clase Observadora.....	51
Tabla 8: Descripción de la Tarea de Ingeniería #1. ....	55
Tabla 9: Descripción de la Tarea de Ingeniería #2. ....	55
Tabla 10: Instrumento de evaluación de la métrica TOC. ....	59
Tabla 11: Instrumento de evaluación de la métrica TOC. ....	59
Tabla 12: Instrumento de evaluación de la métrica RC. ....	61
Tabla 13: Descripción de Caso de prueba de aceptación: Registrar trazas de acción. ....	63
Tabla 14: Descripción de Caso de prueba de aceptación: Registrar trazas de excepción. ....	64



## Introducción

En la actualidad gran parte de la información es considerada confidencial, secreta o con cierto acceso restringido, ya puede ser de carácter gubernamental, militar y empresarial, la que a su vez es almacenada en diversas aplicaciones informáticas a nivel mundial. Dichas aplicaciones deben de presentar además de una buena implementación de su seguridad informática, la capacidad de conocer y monitorear en todo momento la información a la que se está teniendo acceso.

En gran parte de los sistemas informáticos que se desarrollan en la actualidad, se hace necesario un control y registro de los documentos electrónicos y la información a los que se accede, así como el personal que está accediendo a la misma. Además de poder almacenarla de manera rápida y fiable para que esté disponible en cualquier momento que se necesite. Estos elementos favorecen a que la aplicación de software que se desarrolla cumpla con las características para que sea auditable.

Cuba no se queda ajena al proceso de informatización, tarea en la que están inmersos muchos órganos del estado y del país, tal es el caso de la Fiscalía General de República de Cuba, en colaboración con la Universidad de Ciencias Informáticas (UCI), con la tarea de informatizar los principales procesos que se llevan a cabo en el primero de ellos, creándose así el proyecto de Sistema de Informatización de la Gestión de las Fiscalías (SIGEF II) que se encuentra en su Fase II actualmente.

La aplicación en desarrollo en el proyecto Sistema de Informatización de la Gestión de las Fiscalías está siendo elaborada en el marco de trabajo Symfony 1.3, el cual permite su fácil gestión y mantenimiento pero no cuenta con un eficiente control y registro de los eventos, trazas y excepciones que deben ser almacenados y correctamente gestionados según el comportamiento seguido por las herramientas desarrolladas tanto a nivel nacional como internacional, encargadas de manejar la trazabilidad de los sistemas informáticos, requisito fundamental para avalar la competitividad del software, así como satisfacer las necesidades del proyecto y brindar la información necesaria de la manera más asequible para el usuario.

Debido a las características del proyecto, en el cual se maneja información clasificada y confidencial en cuanto a procesos que se llevan a cabo en la Fiscalía, se hace necesario el control de las huellas dejadas por los usuarios a través de los diferentes subsistemas. Las



informaciones y registros que los fiscales y usuarios finales de la aplicación manejan, actualizan o crean no son registradas en ningún momento, tampoco el personal que en ese instante está accediendo a la misma. Además el control de las excepciones que pueden producirse en el sistema y que pueden afectar la estabilidad y funcionamiento del mismo no son correctamente registradas.

El sistema informático actual no reúne los requisitos para ser auditable debido a que no se cuenta con un medio capaz de registrar y llevar el control de las trazas de la aplicación a nivel de excepciones y acciones de un usuario en todo momento.

Dicha situación problemática genera el siguiente **problema a resolver** en la investigación:

¿Cómo contribuir a almacenar de modo fiable y completo los pasos críticos en el manejo de documentos electrónicos y registros del Sistema de Informatización de la Gestión de las Fiscalías de modo que estén disponibles ante posibles eventos de auditoría?

Para el desarrollo de la investigación se define como **objeto de estudio**:

Proceso de desarrollo de software de gestión.

Teniendo como **objetivo general**:

Desarrollar el componente “Gestor de Trazas” para el Sistema de Informatización de la Gestión de las Fiscalías de modo que permita conocer los pasos críticos en el manejo de documentos electrónicos y registros para que estén disponibles ante posibles eventos de auditoría.

Enmarcado en el **campo de acción**:

Proceso de desarrollo de software para la gestión de auditorías de sistemas en el marco de trabajo Symfony 1.3 en el proyecto Sistema de Informatización de la Gestión de las Fiscalías.

La investigación posee como **idea a defender**:

El desarrollo del componente “Gestor de Trazas” para el Sistema de Informatización de la Gestión de las Fiscalías contribuirá a conocer los pasos críticos en el manejo de



documentos electrónicos y registros para que estén disponibles ante posibles eventos de auditoría.

Para la misma se cuentan con los siguientes **objetivos específicos**:

- Realizar el marco teórico de la investigación.
- Realizar el diseño del componente Gestor de Trazas.
- Realizar la implementación del componente Gestor de Trazas.
- Validar la solución propuesta.

Dándole cumplimiento a los objetivos específicos se definen las siguientes **tareas de la investigación**:

- Estudio del estado del arte de la trazabilidad en el marco de trabajo Symfony 1.3.
- Obtención del Modelo de Diseño para el componente Gestor de Trazas.
- Obtención de las tarjetas CRC (Clase, Relación, Colaboración).
- Obtención del Diagrama de Despliegue.
- Diseño para la implementación de casos de pruebas.

Dando cumplimiento a cada objetivo específico mediante las tareas se obtendrá como resultado de la investigación la realización del componente “Gestor de Trazas” en el marco de trabajo de Symfony 1.3 para el proyecto productivo Sistema de Informatización de la Gestión de las Fiscalías.

La investigación estará basada en los siguientes **métodos científicos**:

### **Métodos Teóricos**

- ✓ Analítico - sintético: partiendo de un detallado análisis de toda la información existente sobre el almacenamiento y la gestión de registros y documentos electrónicos mediante las trazas generadas en el marco de trabajo Symfony 1.3, se logrará reunir la información necesaria para referenciar herramientas desarrolladas con los mismos fines.
- ✓ Histórico - Lógico: partiendo de la información analizada sobre la evolución de un registro de trazas a lo largo del sistema y su conformación, se logrará definir los parámetros y aspectos esenciales a tener en cuenta a la hora de realizar el componente.





## Métodos Empíricos

- ✓ Medición: se definirán un conjunto de métricas, con las cuales se obtendrá información cuantitativa sobre el componente.

### **El trabajo está estructurado en tres capítulos de la siguiente forma:**

El **Capítulo 1** se enmarca en la Fundamentación Teórica de la investigación. Para ello se aborda el tema del manejo de los documentos y registros electrónicos en el proyecto productivo Sistema de Informatización de la Gestión de las Fiscalías (SIGEF II) mediante todo un estudio del arte de los procesos realizados en las fiscalías y las formas existentes para el registro de los mismos. Además se describen en este capítulo las principales herramientas, plataforma y metodología usadas.

En el **Capítulo 2** se realiza una descripción de las principales características del componente detallando la propuesta de solución. Se ilustran los principales requisitos, así como los artefactos generados, teniendo en cuenta la metodología seleccionada. Además se diseñan los casos de prueba que contribuirán posteriormente a la validación de la solución.

El **Capítulo 3** aborda las validaciones del sistema describiendo las pruebas unitarias y de aceptación a realizar. Se detallan las métricas utilizadas mostrándose los resultados obtenidos de la aplicación de las mismas.



## Capítulo 1: Fundamentación teórica

En el presente capítulo se explicarán un conjunto de conceptos fundamentales que se utilizarán a lo largo del documento para un mejor entendimiento de la investigación. Se abordarán las características del proyecto Sistema de Informatización de la Gestión de las Fiscalías (SIGEF II), para un mejor entendimiento del ámbito en el cual será desarrollada la propuesta de solución.

Se realizará un estudio sobre las diferentes herramientas que existen en el mundo y en Cuba para la gestión de las trazas, así como el mecanismo que utiliza el marco de trabajo Symfony 1.3 para el control y registro de eventos.

Luego se detallará la propuesta de solución describiendo la arquitectura, metodología y principales herramientas a utilizar, definiendo finalmente las métricas, que permitirán validar el diseño propuesto.

### 1.1 Conceptos Fundamentales

#### Trazas

La traza es la marca dejada por un elemento al haberse desplazado a una nueva posición, por lo que llevándolo al contexto informático se puede decir que las trazas son una colección de eventos y datos que representan el historial o rastro dejado por cierto proceso cuando se ejecuta en un sistema determinado. (1)

#### Auditoría

De una forma muy general, la auditoría es una actividad o acción (o un grupo de estos) realizadas por uno o varios elementos (humanos, máquinas...) con el objetivo de prevenir, detectar o corregir errores, omisiones o irregularidades que afecten al funcionamiento de una actividad o proceso. (2)

#### Auditoría Informática

Es el conjunto de técnica, actividades y procedimientos, destinados a analizar, evaluar, verificar y recomendar en asuntos relativos a la planificación, control, eficacia y seguridad del servicio informático en la empresa. (3)



### Marco de Trabajo o Framework

Es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje para ayudar a desarrollar y unir los diferentes componentes de un proyecto. (4)

### Sistema auditable

Para que un sistema sea auditable, es esencial poder reconstruir los pasos críticos de los procesos por los que sus documentos electrónicos o registros fueron pasando; o por lo menos almacenar de algún modo fiable y completo esta información para que esté disponible ante posibles eventos de auditoría de los sistemas. (30)

## **1.2 Características del proyecto Sistema de Informatización de la Gestión de las Fiscalías (SIGEF II).<sup>1</sup>**

En el proyecto SIGEF II se maneja información de carácter confidencial referente a los procesos fiscales por lo que mantener la seguridad es de vital importancia, ya que la información no puede ser de acceso al público, trayendo consigo que la disponibilidad e integridad de la misma sea una tarea primordial. La solución de software en desarrollo será una aplicación web que comprende los siguientes módulos:

---

<sup>1</sup> Proyecto Técnico 1.2

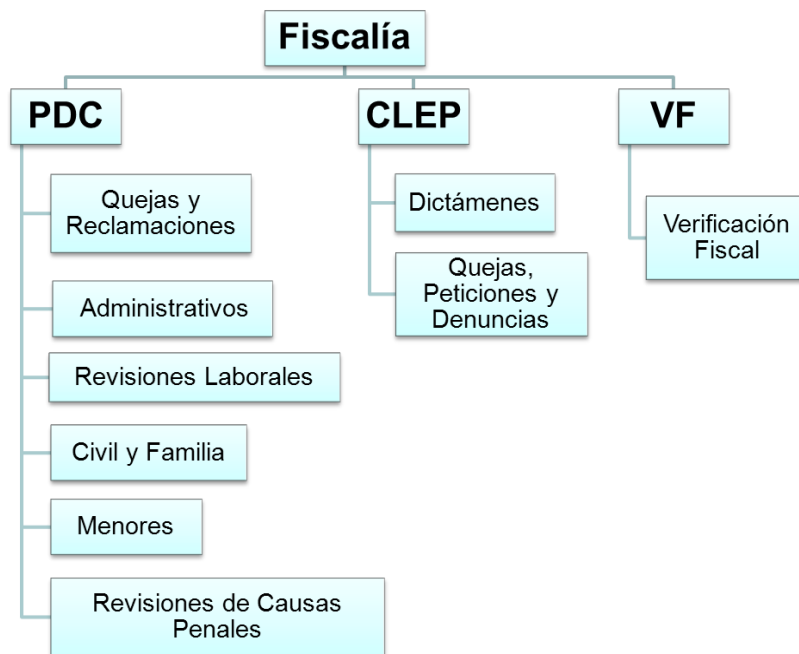


Figura 1: Estructura del proyecto Sistema de Informatización de la Gestión de las Fiscalías.

### 1.2.1 Arquitectura del proyecto

Teniendo en cuenta las características del proyecto se define una propuesta arquitectónica que debe proporcionar flexibilidad, interoperabilidad y robustez, a partir de las necesidades y restricciones identificadas; debido a esto es que se organiza la arquitectura utilizando el estilo arquitectónico en capas.

Se definen como patrones arquitectónicos y de diseño el **MVC** (Modelo Vista Controlador), **Decorator**, **FrontController**, **Data Mapper**.

Las herramientas definidas para el proyecto se han agrupado teniendo en cuenta cada una de las capas o vistas de los patrones arquitectónicos definidos anteriormente, quedando de la siguiente forma:

#### Capa de Presentación o Vista:

- Ajax con JQuery para el manejo de las funciones JavaScript y HTML dinámico.

#### Capa del Modelo:



- ORM Propel para el mapeo de datos.
- BD Postgres 8.4.

**Modelado y diagramas:** Visual Paradigm (3.4).

**Prototipado:** Pencil es la alternativa libre de Axure y multiplataforma para realizar esta tarea. Es una simple extensión de Firefox pero cuenta con una potencia y flexibilidad considerables. Permite crear proyectos con varias pantallas, en las que es posible añadir cualquier elemento (botones, tablas, cajas de activación, deslizadores...).

**Lenguaje de programación:** PHP5 ya que ofrece las siguientes ventajas:

- Es un lenguaje multiplataforma.
- Tiene capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Posee una amplia documentación en su página oficial (<http://www.php.net/>), entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones.

**IDEs:** Zend Studio (7.1.0) y Netbeans (6.9) que permiten una fuerte integración con php y demás herramientas a utilizar en el proyecto.

**Framework o Marco de Trabajo:** Symfony 1.3 que posee amplia cobertura de funciones y compatibilidad para generar sistemas reutilizables, implementa MVC ofreciendo numerosas facilidades para el desarrollo de aplicaciones web.

**Gestión de la Configuración:**

Subversion: Es una herramienta de control de versiones, también puede considerarse como un repositorio que guarda la información de archivos y directorios, incluyendo la información



asociada a las modificaciones realizadas. Es utilizada para mantener actualizada las diferentes modificaciones del código de la aplicación SIGEF II. (31)

Alfresco: Software destinado para la gestión documental y de contenidos, capaz de implementar satisfactoriamente los requisitos de gestión documental dentro de una organización. Permite mantener los documentos generados de la aplicación actualizados y accesibles al equipo de trabajo. (32)

Redmine: Herramienta destinada para la administración de tiempo, evaluación y personal asignado a una tarea específica dentro de un proyecto productivo. Mediante la misma se controlan los diferentes roles, responsabilidades y cumplimiento de las tareas asignadas contribuyendo así a una mejor organización administrativa del proyecto.

### **1.3 Manejo de las trazas a nivel internacional y en la industria cubana del software.**

En la actualidad las organizaciones han incorporado a su funcionamiento las ventajas de las tecnologías de la información, modificando sus procesos de negocio para mejorar el rendimiento y la productividad.

Sin embargo, en el diseño de estos sistemas de información, desde el comienzo, ha primado el rápido funcionamiento y la puesta en marcha de los servicios, sin detenerse a pensar en implementar un mecanismo de control y auditoría sobre los procesos. En la actualidad esas desconsideraciones sobre la importancia de garantizar la fiabilidad y trazabilidad de los procesos que han sido automatizados están poniendo de manifiesto una gran preocupación por la gestión y control de las tecnologías de la información. Como consecuencia de esto las aplicaciones informáticas han implementado sistemas de trazas que contribuyen a verificar que los programas y usuarios realizan exactamente las funciones previstas y no otras, tributando a un requisito primordial de todo sistema: la capacidad de ser auditable.

Las aplicaciones son sometidas constantemente a auditorías informáticas con el objetivo de detectar deficiencias e irregularidades en el uso de los sistemas de información. Las trazas permiten a los auditores informáticos evaluar en alguna medida la confianza que se puede depositar en un sistema basado en la evidencia que brindan las mismas y contribuyen a avalar el sistema en cuanto a seguridad y fiabilidad de la información que se maneja.



De ahí que las aplicaciones implementen un sistema de control de la trazabilidad que responda a sus necesidades y contribuya a garantizar la seguridad para alcanzar un mejor nivel, satisfaciendo las necesidades del entorno que exige calidad y confianza.

#### **1.4 Principales herramientas de registro de trazas.**

En el contexto internacional y nacional se han desarrollado diversas aplicaciones destinadas específicamente a tener un control y registro de las trazas debido a la importancia que las mismas implican, teniendo en cuenta la información que maneja el sistema donde esté integrada la herramienta.

##### **1.4.1 TraceListener**

Básicamente TraceListener es una herramienta de software propietario para el registro de eventos a partir de las trazas generadas por el sistema operativo. Esta herramienta no contempla las acciones individuales de cada usuario y tampoco permite la interoperabilidad o capacidad de integrarse a otra aplicación. Tiene incorporado el registro de trazas a través de los siguientes lenguajes:

- .NET / C # / VB / J #.
- Java.
- C + +.
- Perl.
- PHP.
- PLSQL / Oracle Databaselogging.

Características:

- Capaz de escuchar en los puertos TCP o UDP para el registro de eventos entrantes de aplicaciones.
- Cargar / Guardar y recibir registros de eventos simultáneamente desde múltiples fuentes.
- Vista por detalles de cada registro de eventos.



- Capacidad de búsqueda completa.
- Interfaz de usuario en inglés y alemán.
- Extensión de las posibilidades de filtro (por ejemplo, por fecha).
- Gestor de Base de Datos Oracle.

#### **1.4.2 Audit Logger 5.5**

Los Sistemas de Planificación de Recursos de la Empresa (ERP), son sistemas de gestión de información que integran y automatizan muchas de las prácticas de negocio asociadas con los aspectos operativos o productivos de una empresa.

Audit Logger es una aplicación desarrollada en software propietario para Sage Accpac ERP<sup>2</sup> que proporciona a las organizaciones una herramienta de auditoría, la cual registra las acciones del usuario, las alteraciones de seguridad y las modificaciones de datos. Audit Logger es un sistema importante para proporcionar una solución completa de control interno dedicado específicamente a las nóminas de sueldos, el mismo no permite integración con otros sistemas que no sean propios de Sage Accpac ERP.

Características:

- Proporciona características que permiten gestionar auditorías.
- Permite el registro de adiciones, cambios y supresiones que un usuario realice en el sistema.
- Los datos de la auditoría pueden ser impresos.
- Registra la información importante de la PC del cliente, tales como dirección IP y el usuario de Windows autenticado.
- Los informes pueden ser filtrados por rango de vista, el usuario o fecha y puede ser limitado a los campos que se desean en la auditoría.
- Interfaz en idioma Inglés.

---

<sup>2</sup> Poderoso Software de Gestión Empresarial





### **1.4.3 CRM Logger**

Las herramientas de gestión de relaciones con los clientes (Customer Relationship Management CRM) son las soluciones tecnológicas para conseguir desarrollar la "teoría" del marketing relacional el cual consiste en generar relaciones rentables con sus clientes, en base al estudio del comportamiento de los compradores.

CRM Logger es una aplicación desarrollada también para sistemas ERP, debido a la necesidad de que poseen los mismos de ser informados de su situación en cuanto a las entradas periódicas de los usuarios al sistema. Además, de poder realizar un seguimiento de las transacciones importantes como la creación de usuarios, órdenes de venta y cancelación de facturas. El desarrollo de la herramienta está basado en software propietario como: Windows 2008 Server, Visual Studio 2008, SQL Server 2005 y el mismo no permite la integración con otra aplicación que no sea desarrollada con las mismas características.

Características:

- Cada traza posee un identificador de un servicio y tal vez una categoría. Para que sea más fácil de filtrar los registros específicos.
- Cada Traza contempla diferentes acciones del usuario como insertar, actualizar y eliminar comunicado. Además posee una especificación de su tipo como (advertencia, error, depuración).
- Interfaz en idioma Inglés.

### **1.4.4 Acaxia para el marco de trabajo SAUXE**

Sauxe es un Marco de Trabajo que contiene un conjunto de componentes reutilizables que provee una estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. El mismo fue desarrollado en la UCI como marco de trabajo para el desarrollo de sistemas ERP.

Sauxe posee entre sus componentes la herramienta Acaxia que se encarga del registro y control de las trazas en el sistema, la cual fue desarrollada según el paradigma MVC (modelo-vista-controlador).



La integración de la herramienta Acaxia a otra aplicación trae consigo una reestructuración de la arquitectura del sistema al cual será integrada, que incide en la redistribución de parte del código del sistema a integrar

Entre sus características se encuentran:

- Desarrollada usando PHP 5, Marco de Trabajo Zend (ZendFramework), Doctrine y ExtJs.
- El Marco de Trabajo Sauxe utiliza programación orientada a aspectos, que permite una adecuada modularización de la aplicación y posibilita una mejor separación de los diferentes conceptos.
- Registro de trazas según URL, Autenticación, Cierre de sesión, Acción, Excepción, Excepción de integración, IoC, Rendimiento, Integración y Datos.
- Cuenta con interfaz gráfica que permite configurar, gestionar los tipos de trazas, eliminar las trazas.
- Permite realizar reportes exportándolas en formato XML.
- Permite filtrar las trazas según criterios como: Fecha, Categoría y Tipo.

### 1.5 Mecanismo de control de las trazas de Symfony

Luego del análisis de las diferentes herramientas existentes para el control de las trazas se realizó un estudio del mecanismo que brinda el Marco de Trabajo Symfony 1.3 para el registro de las mismas mediante los logs.

El log es un historial que informa lo que fue cambiado y cómo se realizó (referente a la información). El log permite analizar cronológicamente que es lo que ha sucedido con la información que está en el sistema o que existe dentro de la base de datos. (18).

El marco de Trabajo Symfony 1.3 gestiona las trazas a través de los archivos de log. Entre los directorios de la raíz de proyectos de Symfony 1.3 se encuentra log, en el cual se guarda todos los archivos generados por Symfony 1.3. También se puede utilizar para



guardar los logs del servidor web, de la base de datos o de cualquier otro componente del proyecto. Symfony 1.3 crea un archivo de log por cada aplicación y por cada entorno.

En el archivo de configuración php.ini, se especifican los eventos PHP que se guardan en el archivo de log

La sintaxis de los archivos de log generados es muy sencilla, cada evento resulta en una nueva línea en el archivo de log de la aplicación, dicha línea incluye la fecha y hora en la que se ha producido, el tipo de evento, el objeto que ha sido procesado y otros detalles relevantes que dependen de cada tipo de evento y/o objeto procesado.

```
Nov 15 16:30:25 symfony [info ] {sfAction} call "barActions->executemessages()"
Nov 15 16:30:25 symfony [info ] {sfPropelLogger} executeQuery: SELECT bd_message.ID...
Nov 15 16:30:25 symfony [info ] {sfView} set slot "leftbar" (bar/index)
Nov 15 16:30:25 symfony [info ] {sfView} set slot "messageblock" (bar/mes...
Nov 15 16:30:25 symfony [info ] {sfView} execute view for template "messa...
Nov 15 16:30:25 symfony [info ] {sfView} render "/home/production/myproject/...
Nov 15 16:30:25 symfony [info ] {sfView} render to client
```

Figura 2: Fragmento del contenido de un log de Symfony 1.3

Symfony 1.3 define ocho niveles diferentes para los mensajes de log: **emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info** y **debug**. El mecanismo encargado de gestionar los logs es la clase abstracta **sfLogger**, cualquier logger en Symfony 1.3 deberá heredar de la misma. Entre sus métodos, uno de los más importantes es **sfLogger: doLog (\$message, \$priority)** que recibe dos parámetros, el mensaje y la prioridad.

Además Symfony 1.3 posee una barra de depuración web que permite acceder directamente a los eventos guardados en el log entre otras opciones de configuración que posee. En la barra se permite desplegar los detalles de las consultas realizadas a la base de datos y una tabla con los tiempos empleados en cada elemento de la petición así como excepciones que ocurran. (18).

El principal problema radica en que los datos se muestran sin tener en cuenta al usuario que en ese momento se registra, ni sus acciones individualmente, ya que los datos se muestran de forma generalizada. Además la estructura de la información no permite tener una idea concreta de los datos a conservar y registrar de la aplicación en desarrollo. Tampoco cuenta con filtros que permitan gestionar las trazas según criterios como: fecha, usuario, tipo de traza, entre otros.

**Capítulo 1: Fundamentación teórica**

Luego del estudio de las herramientas y el mecanismo utilizado por Symfony 1.3 para la gestión de las trazas se ilustra mediante una tabla comparativa, teniendo en cuenta las características recopiladas de las mismas, los aspectos negativos que no facilitan su integración a la aplicación en desarrollo SIGEF II, debido a que no satisfacen completamente las necesidades existentes.

Herramientas	Base de Datos	Idioma	Tipo software	Acciones de usuarios	Filtrado de parámetros	Interoperabilidad
<b>TraceListener</b>	<i>Oracle</i>	<i>Inglés y Alemán</i>	<i>Software propietario</i>	<i>NO</i>	<i>Parcial</i>	<i>NO</i>
<b>Audit Logger 5.5</b>	<i>Microsoft SQL Server</i>	<i>Inglés</i>	<i>Software propietario</i>	<i>SI</i>	<i>SI</i>	<i>NO</i>
<b>CRM Logger</b>	<i>SQL Server 2005</i>	<i>Inglés</i>	<i>Software propietario</i>	<i>SI</i>	<i>Parcial</i>	<i>NO</i>
<b>Acaxia</b>	<i>PostgreSQL 8.3</i>	<i>Español</i>	<i>Software libre</i>	<i>SI</i>	<i>SI</i>	<i>NO</i>
<b>Mecanismo de Symfony</b>	<i>---</i>	<i>Español</i>	<i>Software libre</i>	<i>NO</i>	<i>NO</i>	<i>SI</i>

*Tabla 1: Análisis comparativos de las herramientas de trazas.*

En el caso de la herramienta Acaxia, cumple con casi todos los parámetros para ser integrada al proyecto, pero hacerlo traería consigo una reestructuración de la aplicación SIGEF II, dividiendo la misma en subsistemas y refactorizando su código casi completamente para su integración. Tomando en consideración la etapa de implementación en la que se encuentra sería riesgoso, además la fase en desarrollo del proyecto tuvo una versión anterior la que está relacionada con la misma, desplegada ya en las fiscalías, compuesta por un conjunto de funcionalidades que tendrían que separarse trayendo consigo una re-implementación de su código también, lo que no es posible en estos momentos debido a las condiciones antes expuestas.

**1.6 Propuesta de solución**



Teniendo en cuenta lo antes mencionado se hace necesario el desarrollo de un componente auxiliándose del mecanismo que brinda Symfony 1.3 para el control de las trazas, incorporándole además otros elementos y parámetros como el registro de las excepciones, las consultas y acciones de cada usuario a través de la aplicación, que permita mostrar de forma más organizada la información de dichos eventos en el momento de su consulta, la que debe ser visualizada de una manera entendible por los usuarios finales del software.

Todas estas características antes expuestas son las bases para el desarrollo del componente “Gestor de Trazas” y su integración al sistema SIGEF II.

A continuación se abordará sobre la metodología seleccionada, así como las herramientas y tecnologías definidas para el desarrollo del componente teniendo en cuenta características fundamentales que se ajusten al entorno de desarrollo.

### **1.6.1 Metodología de desarrollo de software**

En la actualidad no está definida una metodología universal que se aplique a todos los proyectos, la selección de la misma depende de las características específicas que tenga cada uno de estos. (6)

Se podrían clasificar en dos grandes grupos:

**Metodologías Pesadas:** orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas y notaciones a utilizar.

**Metodologías ligeras/ágiles:** orientadas a la interacción con el cliente y el desarrollo incremental del software, mostrando versiones parcialmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando.

#### Metodologías Pesadas.

Son las más tradicionales, se centran en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar, y requiere una extensa documentación, ya que pretende predecir todo de antemano. Una de las metodologías pesadas más conocidas y utilizadas es la Metodología RUP (Rational Unified Process).

Otras metodologías que se pueden encontrar son:

- MSF (Microsoft Solution Framework).
- Win-Win Spiral Model.



- Iconix.

### Metodologías ligeras/ágiles.

Principales ideas de la metodología ágil:

- Se encarga de valorar al individuo y las relaciones del equipo de desarrollo más que a las herramientas o los procesos utilizados.
- Se hace más importante crear un producto de software que funcione, que escribir mucha documentación.
- El cliente está en todo momento colaborando en el proyecto.
- Es más importante la capacidad de respuesta ante un cambio realizado que el seguimiento estricto de un plan.

Una de las metodologías ágiles más conocidas y utilizadas es **Programación Extrema o XP (EXTREME PROGRAMMING)**.

Otras metodologías ágiles:

- SCRUM.
- Crystal Methodologies.
- Dynamic Systems Development Method (DSDM).
- Adaptive Software Development (ASD).
- Feature Driven Development (FDD).
- Lean Development (LD).

En el desarrollo del componente se utilizará la metodología **XP** debido a que es empleada en proyectos de equipos pequeños y poco tiempo de entrega, teniendo como premisa que el funcionamiento del software es más importante que la documentación exhaustiva que se genere, lo que es muy acertado en el desarrollo del mismo.

La programación extrema se basa en la simplicidad, la comunicación y el reciclado del código, aspectos fundamentales a tener en cuenta para el desarrollo. XP es muy acertada ya que es ligera, mantiene una fuerte comunicación con el cliente, es basada en la programación por parejas y pobre en cuanto a documentación, lo que en el caso de proyectos que tengan un plazo corto de entrega constituye una ventaja.

El ciclo de vida ideal de XP consiste en cuatro fases fundamentales: Planificación, Diseño, Desarrollo y Prueba. (7)

Se trata de realizar ciclos de desarrollo cortos (llamados iteraciones), con entregables funcionales al finalizar cada ciclo. En cada iteración se realiza un ciclo completo de análisis,



diseño, desarrollo y pruebas, pero utilizando un conjunto de reglas y prácticas que caracterizan a XP.

Esta metodología divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida, diseñadas por el cliente final.

La figura ilustra el ciclo de vida de XP:

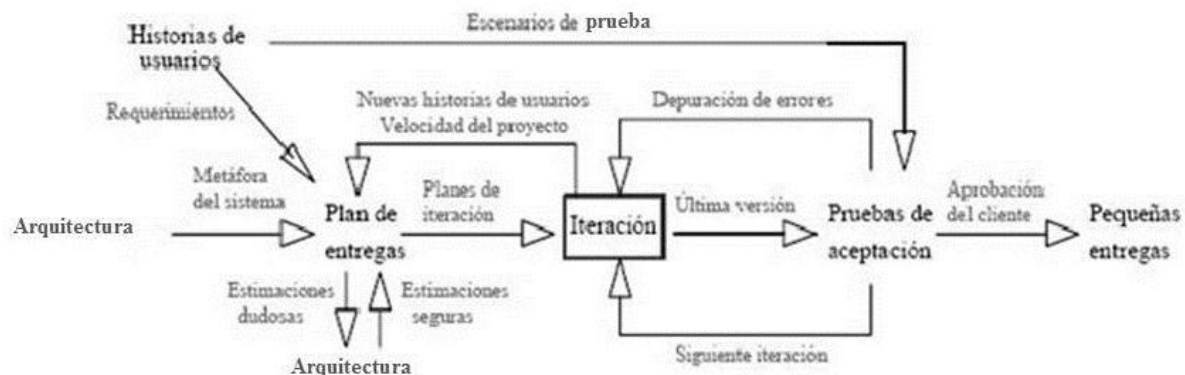


Figura 3: Ciclo completo de XP

### 1.6.2 Arquitectura del componente

Luego de seleccionada la metodología de desarrollo a seguir se define la arquitectura del componente especificando un estilo arquitectónico en capas utilizando el patrón modelo vista controlador (MVC).

#### Modelo Vista Controlador

MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Su finalidad es mejorar la reusabilidad por medio del desacople entre la vista y el modelo. (8)

Función de cada capa del patrón:

**Modelo:** representa la información con la que trabaja la aplicación y se encarga de acceder a los datos.



Entre otras funciones se encuentran:

- Define las reglas de negocio (la funcionalidad del sistema).
- Lleva un registro de las vistas y controladores del sistema.
- Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo.

**Vista:** transforma la información obtenida por el modelo en las páginas web a las que acceden los usuarios.

Entre otras funciones se encuentran:

- Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
- Pueden dar el servicio de actualización, para que sea invocado por el controlador o por el modelo.

**Controlador:** es el encargado de coordinar todos los demás elementos y transformar las peticiones del usuario en operaciones sobre el modelo y la vista.

Entre otras funciones se encuentran:

- Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).

Para el desarrollo del componente la utilización de este patrón proporciona un conjunto de ventajas como la posibilidad de poseer diferentes vistas para un mismo modelo, además de construir nuevas vistas sin necesidad de modificar el modelo subyacente desarrollado. También proporciona un mecanismo de configuración permitiendo que el modelo pueda verse como una representación estructurada.

### ***1.6.3 Herramientas y tecnologías definidas para la realización del componente:***

Luego de definida la arquitectura a utilizar en el desarrollo del componente se describen las herramientas y tecnologías a utilizar para su implementación.





### 1.6.3.1 Visual Paradigm

Utilizada para el modelado y diagramas, facilita la automatización del ciclo de vida de desarrollo. Es una herramienta colaborativa, es decir permite a múltiples usuarios trabajar sobre el mismo proyecto así como obtener una mejor calidad en el producto. Es fácil de instalar y actualizar, y compatible entre ediciones. (9)

Entre sus ventajas se encuentran:

- Automatiza la realización de diagramas.
- Genera estructura de código.
- Ayuda a la creación de relaciones en la base de datos.
- Ayuda a la documentación del sistema.

Visual Paradigm es una herramienta CASE<sup>3</sup> que utiliza UML como lenguaje de modelado, ha sido creado para todas las personas que estén interesadas en el diseño de software orientado a objetos. Esta herramienta facilita la creación de diferentes diagramas UML y posibilita generar código a partir de diagramas así como generar documentación.

Entre sus principales características se encuentran:

- Soporta la última notación UML 2.1.
- Editor de detalles de casos de uso.
- Generación de bases de datos.
- Generador de informes para generación de documentación.
- Multiplataforma.
- Provee soporte para la generación de código PHP.

### 1.6.3.2 Ajax con JQuery

---

<sup>3</sup> CASE(Computer Aided Software Engineering): Estas herramientas brindan ayuda en el ciclo de desarrollo de un software



AJAX (Asynchronous JavaScript and XML.) es el arte de intercambiar datos con un servidor, y actualizar las partes de una página web sin necesidad de recargar toda la página. Se utilizará con jQuery que es una librería JavaScript la cual posee características potentes como selectores de CSS, con el poder de crear variables y funciones que interactúen con el documento, siendo fácil de usar, y a su vez muy liviana. Además trae incorporadas funciones adicionales, como manipulación del DOM y detección de navegadores (10), (11).

#### 1.6.3.3 Propel

**Propel** es un ORM (Object-Relational Mapping) para PHP que facilita la labor de desarrollo de aplicaciones web, gracias a la capa que transforma el tratamiento de la BD mediante objetos, con la que se puede recuperar, insertar y modificar datos.

Con la utilización de este ORM se logra rapidez en el desarrollo y abstracción de la base de datos ya que integra un lenguaje propio para realizar las consultas, lo que facilita que los usuarios dejen de utilizar las sentencias SQL y utilicen el lenguaje propio de la herramienta orientado a objetos. (12)

#### 1.6.3.4 Gestor de Base de Datos

##### **PostgreSQL 8.4**

Es un gestor de bases de datos relacional, libre y gratuito. Contiene bloques de código que se ejecutan en el servidor y pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos brinda; desde las operaciones básicas de programación, tales como bifurcaciones y bucles, hasta las complejidades de la programación orientada a objetos o la programación funcional. (13).

Presenta las siguientes propiedades:

- **Atomicidad:** Asegura la realización de una operación, por lo que ante un fallo del sistema esta no queda a medias.
- **Consistencia:** Posibilita la ejecución de aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos.



- Aislamiento: Mediante un componente denominado MVCC (Acceso concurrente multiversión) asegura que una operación no pueda afectar a otras, de esta manera dos transacciones sobre la misma información no genera error.

#### **Versión 8.4**

Entre las mejoras más populares se encuentran:

- Privilegios por columna, que permiten un control más granular de datos confidenciales.
- Nuevas herramientas de monitoreo de consultas que le otorgan a los administradores mayor información sobre la actividad del sistema.

#### 1.6.3.5 Prototipado

##### **Axure**

Axure es una poderosa herramienta para transformar los requerimientos en una visualización detallada del trabajo para apoyar la implementación y reducir el riesgo del proyecto. Es utilizada en creación de prototipos y especificaciones muy precisas para páginas web. (33)

La realización de estos prototipos es de forma interactiva, es decir, se pueden simular muchas de las acciones y navegación que el usuario puede realizar en el nuevo sitio web lo que les dará una visión de cómo el sistema debe funcionar. (34)

#### 1.6.3.6 Lenguaje de Programación

##### **PHP 5**

Es un lenguaje de interpretado de alto nivel, habitualmente se usa para el desarrollo de aplicaciones web.

Es uno de los más utilizados en la actualidad y entre sus principales características se encuentran:

- Es multiplataforma.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- Biblioteca nativa de funciones.
- No requiere definición de tipos de variables.



El principal objetivo de PHP5 ha sido mejorar los mecanismos de POO (Programación Orientada a Objeto) para solucionar las carencias de las anteriores versiones. Un paso necesario para conseguir que PHP sea un lenguaje apto para todo tipo de aplicaciones y entornos, incluso los más exigentes. (16)

#### 1.6.3.7 IDE de desarrollo

##### **NetBeans 6.9**

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es una aplicación informática destinada a los programadores que incluye un número de herramientas que le facilita el trabajo a los desarrolladores, tales como un editor de texto, un compilador, depuradores de código, administración de proyectos, integración con sistemas controladores de versiones o repositorios. NetBeans es un IDE de código abierto, libre y gratuito pensado para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. (14)

#### 1.6.3.8 Marco de trabajo

##### **Symfony 1.3**

Symfony es un marco de trabajo multiplataforma que simplifica el desarrollo de una aplicación web mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Está basado en el clásico patrón de diseño web conocido como arquitectura MVC (Modelo-Vista-Controlador). Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. (15)

Symfony 1.3 está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony 1.3 es compatible con la mayoría de gestores de bases de datos como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft.

##### **Versión 1.3**



Symfony 1.3 es la versión que se debe utilizar para actualizar un proyecto de una versión anterior de Symfony (1.0, 1.1, o 1.2). Esta versión dispone de una capa de retro-compatibilidad que hace que todas las características que se han declarado obsoletas sigan estando disponibles. Por tanto, la actualización suele ser más sencilla, fácil y segura. (17)

Entre las nuevas características que tiene esta versión están:

- Symfony 1.3 es compatible con PHP 5.2.4 o superior.
- La protección frente a los ataques XSS y CSRF ahora es obligatoria para las aplicaciones porque ya están activadas por defecto.
- La validación de los campos de los formularios es más sencilla.
- La barra de depuración web puede ser configurable por los desarrolladores según sus necesidades.
- Se elimina el filtro común, que es el encargado de añadir automáticamente las hojas de estilos y los archivos JavaScript en las páginas. Por lo que es obligatorio incluir manualmente los archivos CSS y JavaScript. Esto mejora el funcionamiento de la aplicación haciéndola más flexible y sencilla permitiendo tener un control preciso sobre dónde se añaden.

Con todas estas características Symfony 1.3 brinda las funcionalidades para crear aplicaciones seguras, flexibles y de manera rápida. (18)

## 1.7 Métricas

Una vez definidas las herramientas a utilizar para el desarrollo del componente se seleccionan las métricas que constituyen un apoyo para conocer mejor el diseño y la construcción de un software. Son utilizadas para crear productos de mejor calidad teniendo en cuenta diferentes parámetros a evaluar en una aplicación siguiendo un conjunto de reglas definidas con anterioridad. (19), (20)

Dentro de las métricas para evaluar el diseño se encuentran el Tamaño Operacional de Clases (TOC) y Relaciones entre Clases (RC) las que serán detalladas a continuación.

### 1.7.1 Métricas para el diseño

Ayudan a evaluar los modelos de análisis y diseño, ofrecen una indicación de la complejidad y del código fuente, así como facilitan el diseño de pruebas más efectivas. (20)

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

#### 1.7.1.1 Tamaño Operacional de Clase (TOC):

Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- **Responsabilidad:** Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
- **Complejidad de implementación:** Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización:** Un aumento del TOC implica una disminución del grado de reutilización de la clase. (21)

#### 1.7.1.2 Relaciones entre clases (RC):



Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

- **Acoplamiento:** Un aumento de las RC implica un aumento del Acoplamiento de la clase.
- **Complejidad de mantenimiento:** Un aumento de las RC implica un aumento de la complejidad del mantenimiento de la clase.
- **Reutilización:** Un aumento de las RC implica una disminución en el grado de reutilización de la clase.
- **Cantidad de pruebas:** Un aumento de las RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase. (21)

### 1.8 Conclusiones Parciales

En este capítulo se realizó un estudio detallado sobre las trazas así como las herramientas existentes en el mundo para el control de las mismas, analizando la metodología de software, así como las herramientas, tecnologías y lenguajes para llevar a cabo el proceso de desarrollo del componente.

Luego de agrupar los elementos necesarios se llega a las siguientes conclusiones:

- El análisis de las soluciones existentes en el mundo debido a sus características no pueden ser integradas al proyecto SIGEF II por lo que se hace necesario desarrollar un componente para ello.
- Las herramientas, tecnologías y metodologías que serán utilizadas para la implementación del componente facilitarán el trabajo en cada etapa de desarrollo.
- Las métricas a utilizar permitirán tener una medida de la calidad del diseño teniendo en cuenta un conjunto de atributos necesarios para evaluar un software como la reutilización de las clases, el acoplamiento, entre otros.



## Capítulo 2: Características del Sistema.

En el presente capítulo se describirán características fundamentales del componente, teniendo en cuenta un conjunto de conceptos. Se definirán los requerimientos no funcionales, así como los funcionales, basándose en las restricciones y características específicas de la solución. Se realizará una descripción del proceso de desarrollo del componente teniendo en cuenta las fases según la metodología XP y los artefactos generados en cada una de ellas.

Conceptos fundamentales:

Usuario: una persona, programa o computadora que utiliza determinado hardware y/o software, mediante el cual obtiene un servicio.

Evento: acción que realiza un usuario dentro de un sistema determinado en un período de tiempo establecido.

Base de Datos: lugar o espacio físico para almacenar datos.

### 2.1 Especificación de los requerimientos del software.

La captura de requerimientos cumple un papel primordial en el proceso de desarrollo de software, se basa en la comunicación entre los desarrolladores, clientes y usuarios, para definir el nuevo sistema que brinde una solución al problema. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema. (35)

Sentando las bases para el desarrollo del componente se definen los requerimientos fundamentales que debe poseer el mismo para satisfacer las necesidades del proyecto.

#### 2.1.1 *Requerimientos no funcionales*

Los requerimientos no funcionales son capacidades o condiciones que todo sistema debe cumplir. Los requerimientos definidos para la realización de esta investigación vienen dados





## Capítulo 2: Características del Sistema

de los estándares mundiales existentes los cuáles son necesarios recoger para la realización de aplicaciones auditables. (22)

### *Confiability*

El sistema debe proveer los mecanismos necesarios para el restablecimiento ante fallos de comunicación u otros, debe crearse un respaldo de electricidad y especificación de tiempo de puesta en marcha nuevamente del componente.

### *Seguridad*

El componente no presentará un sistema de autenticación debido a que el mismo se vinculará al producto SIGEF II el cual ya lo tiene implementado. Al componente tendrán acceso sólo las personas autorizadas para hacer uso del mismo.

*Confidencialidad:* La información que se maneje en el componente estará protegida de acceso no autorizado y divulgación, a partir de los diferentes roles de los usuarios que utilicen el mismo.

- *Integridad:* La información manejada por el componente será objeto de protección contra corrupción, de igual manera el origen y autoridad de los datos debido a la importancia que tienen los mismos.
- *Disponibilidad:* La información debe de encontrarse disponible en todo momento para aquellos usuarios autorizados a acceder a cualquier información en el sistema.

### *Portabilidad*

El componente será multiplataforma y debe ser compatible con los sistemas operativos, Windows y Linux.

### *Soporte*

Se utilizará como sistema gestor de Base de Datos PostgreSQL y como servidor web Apache para el trabajo con PHP y algunas otras herramientas necesarias.

### *Rendimiento*

Los tiempos de respuesta del componente deben ser rápidos así como la velocidad de procesamiento de la información.



### *Usabilidad*

El componente podrá ser usado por personas capacitadas con los conocimientos necesarios para interactuar con el mismo.

### *Políticos-culturales*

El componente podrá ser usado en territorio nacional y estará configurado en el idioma español solamente.

### *Legales*

El componente se debe ajustar y regir por las leyes, decretos y resoluciones establecidas mencionadas en el proyecto técnico de SIGEF II. (2)

#### **2.1.2 Requerimientos funcionales:**

- Registrar trazas de autenticación.
- Registrar trazas de cierre de sesión.
- Registrar trazas de acción.
- Registrar trazas de excepción.
- Mostrar detalles de la traza.
- Buscar trazas.
- Búsqueda Avanzada.
- Imprimir traza.

Los requerimientos funcionales en la metodología XP son traducidos a Historias de Usuario, lo cual constituye el primer artefacto que se genera en dicha metodología.

## **2.2 Etapa de Planificación**



## Capítulo 2: Características del Sistema

Luego de definidos los requerimientos de software se pasa a planificar el desarrollo del componente, en esta etapa se identifican las historias de usuarios, así como el plan de iteraciones y el plan de entregas.

A continuación se ejemplifican las historias de usuarios más representativas, así como el plan de iteraciones y el plan de entrega correspondientes a cada una de ellas.

### 2.2.1 Historias de Usuario

Las historias de usuario son utilizadas para la especificación de los requisitos del software. En ellas se describen brevemente las características que el sistema debe poseer. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento pueden ser reemplazadas por otras más específicas o generales, además de ser modificadas o añadidas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan ser capaces de implementarla en unas semanas. (36)

Historia de Usuario	
<b>Número:</b> 3	<b>Nombre:</b> Registrar trazas de acción.
<b>Usuario:</b> Persona.	
<b>Modificación de Historia</b> <b>Número:</b> 3.	<b>Iteración Asignada:</b> 1.
<b>Prioridad en Negocio:</b> Alta. (Alta / Media / Baja)	<b>Puntos Estimados:</b> N/A.
<b>Riesgo en Desarrollo:</b> Alto. (Alto / Medio / Bajo)	<b>Puntos Reales:</b> N/A.



<p><b>Descripción:</b></p> <p>Inicia cuando se realiza cualquier evento en el sistema que utiliza Symfony 1.3, por ejemplo: consulta, modificación, eliminación y creación, creándose una traza de acción con la acción realizada dentro del sistema por la persona.</p>
--

Tabla 2: Representación de la Historia de Usuario #3.

<b>Historia de Usuario</b>	
<b>Número:</b> 4.	<b>Nombre:</b> Registrar trazas de excepción.
<b>Usuario:</b> Persona	
<b>Modificación de Historia Número:</b> 4.	<b>Iteración Asignada:</b> 2.
<b>Prioridad en Negocio:</b> Alta. (Alta / Media / Baja)	<b>Puntos Estimados:</b> N/A.
<b>Riesgo en Desarrollo:</b> Alto. (Alto / Medio / Bajo)	<b>Puntos Reales:</b> N/A.
<p><b>Descripción:</b></p> <p>Inicia cuando ocurre algún error del sistema (un campo lleno incorrectamente, o un error de programación). Se crea una traza de excepción con los detalles de la misma (mensaje, descripción).</p>	

Tabla 3: Representación de la Historia de Usuario #4.

### 2.2.2 Plan de iteración.

En este paso se dividen las historias de usuarios en tareas que se les asignarán a los programadores las que deben desarrollar en un plazo determinando de tiempo o en varias iteraciones y así poder tener un control del trabajo realizado.

#### Historia de Usuario divididas en tareas.



## Capítulo 2: Características del Sistema

Historia de Usuario	Tareas
Registrar trazas de autenticación.	<ul style="list-style-type: none"><li>- Almacenar los datos.</li><li>- Crear traza.</li></ul>
Registrar trazas de cierre de sesión.	<ul style="list-style-type: none"><li>- Almacenar los datos.</li><li>- Crear traza.</li></ul>
Registrar trazas de acción.	<ul style="list-style-type: none"><li>- Capturar la acción realizada.</li><li>- Almacenar los datos (tipo de acción).</li><li>- Crear traza.</li></ul>
Registrar trazas de excepción.	<ul style="list-style-type: none"><li>- Almacenar datos de la excepción.</li><li>- Almacenar detalles (mensaje, descripción) de la excepción.</li><li>- Crear traza.</li></ul>
Mostrar detalles de la traza	<ul style="list-style-type: none"><li>- Buscar datos de la traza seleccionada.</li><li>- Mostrar datos.</li></ul>
Buscar trazas.	<ul style="list-style-type: none"><li>- Manejar los criterios de búsqueda introducidos (rango de fecha).</li><li>- Mostrar traza de datos coincidentes con los criterios.</li></ul>
Búsqueda avanzada.	<ul style="list-style-type: none"><li>- Manejar los criterios de búsqueda introducidos (rango de fecha desde hasta, usuario, módulo, tipo, subsistema).</li><li>- Mostrar traza de datos coincidentes con los criterios.</li></ul>
Imprimir traza.	<ul style="list-style-type: none"><li>- Generar plantilla a imprimir de acuerdo a la traza seleccionada.</li></ul>

Tabla 4: Distribución de las tareas por cada Historia de Usuarios.

Para que exista un equilibrio entre las diferentes secuencias de trabajo se procede a dividir el proyecto en 3 iteraciones que se describen a continuación, realizándose un balance entre



ellas tributando a que cada una sea escenario de un conjunto de historias de usuarios que demandarán un desempeño aproximado del equipo de trabajo.

Se agruparon las diferentes historias de usuario en cada una de las siguientes iteraciones.

### Iteración 1

Durante la primera iteración se realizarán las historias de usuario registrar trazas de autenticación y registrar trazas de acción, las que son claves en la solución. Finalizada esta iteración se tendrá un mejor entendimiento del componente al quedar concluida la tarea de registrar trazas de acción requisito clave que debe de tener el producto final.

### Iteración 2

En el transcurso de esta iteración se realizarán las historias de usuario registrar trazas de cierre de sesión y registrar trazas de excepción que es una de las más complejas a implementar por el equipo de desarrolladores, la que requiere el mayor esfuerzo posible por parte de los mismos.

### Iteración 3

Es la última iteración en la que se realizan las historias de usuarios referentes a mostrar detalles de la traza, las búsquedas y finalmente el imprimir. Terminada esta iteración quedarán implementadas en su totalidad las historias de usuario tributando al desarrollo del componente en su totalidad.

La siguiente figura ilustra la representación de la primera iteración descrita anteriormente, así como el conjunto de tareas definidas para cada historia de usuario y el programador responsable de realizarlas.



Plan de iteración #1

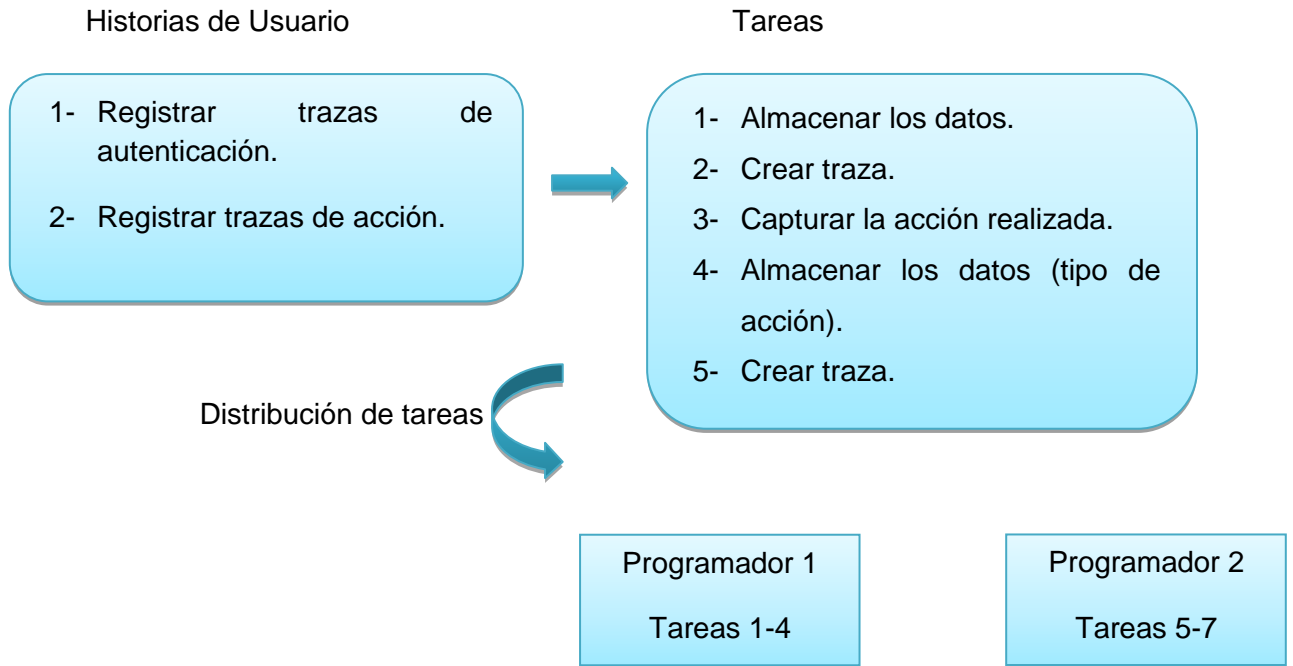


Figura 4: Descripción del plan iteración #1.

**2.2.3 Plan de entregas**

Es un artefacto generado como parte de la planificación que permite realizar una organización del tiempo de las entregas, marcando el final de cada una de las iteraciones permitiéndoles así a los programadores una guía para el desarrollo en tiempo de la solución.

Artefacto	Hito	Entrega
Gestor de Trazas	Final de la primera iteración	24-27 febrero 2012
Gestor de Trazas	Final de la segunda iteración	1-5 abril 2012
Gestor de Trazas	Final de la tercera iteración	27 abril 2012

Tabla 5: Plan de entregas de las iteraciones.



## 2.3 Etapa de Diseño

Una vez planificada la realización del componente se procede al diseño del mismo, definiendo los patrones a utilizar. También se obtienen las tarjetas CRC (Clase, Cargo, Colaboración) donde se ilustran las clases y la información necesaria a almacenar en cada una de ellas. Se define el modelo de datos mostrando las entidades persistentes así como sus relaciones y finalmente se obtiene el diagrama de despliegue para visualizar el entorno en el cual estará desplegado el componente una vez finalizada su implementación.

### 2.3.1 Patrones de Diseño

Los patrones de diseño son un conjunto de estrategias, o buenas prácticas, que pueden facilitar el trabajo en muchas situaciones en el momento de realizar una aplicación orientada a objetos. (23)

Para el desarrollo del componente se utilizaron un conjunto de patrones que se explican a continuación:

Decorator (Decorador): Facilita la agregación de funcionalidades dinámicamente a las clases. Este patrón viene implícito en el marco de trabajo Symfony 1.3, el cual permitió a través del archivo layout.php, que también se denomina plantilla global, almacenar el código HTML que es común a todas las páginas del componente, para no tener que repetirlo en cada una de ellas. El contenido de la plantilla se integra en el Layout (plantilla) decorando la misma.

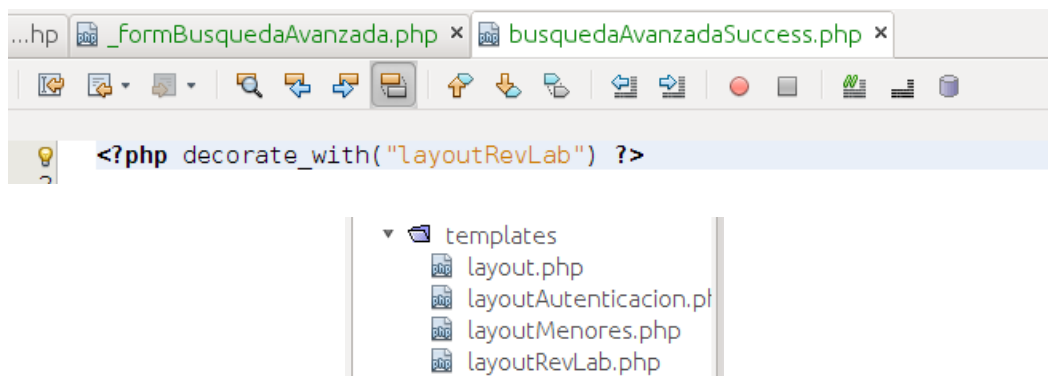


Figura 5: Ejemplo de uso del patrón Decorator en el componente.

Observer (Observador): Se utilizó en el momento de capturar los eventos generados por las

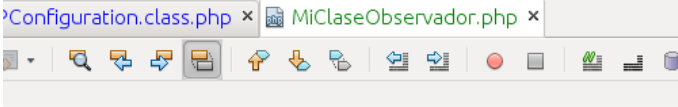




acciones y excepciones en el sistema, facilitando de esta manera poder agruparlos según los parámetros necesarios para su posterior manejo. Este patrón define una dependencia de uno a muchos entre clases u objetos, de tal forma notifica y actualiza, de manera automática, a todas sus dependencias cuando un objeto cambia de estado. Observer notifica a un número determinado de Subscribers (subscriptores) acerca de los cambios de su estado.

```
class CLEPConfiguration extends sfApplicationConfiguration
{
    public function configure()
    {
        //--captura de excepciones--
        $this->dispatcher->connect(
            'CLEP.excepcion', array('MiClaseObservador','errores'));
        //captura de eventos---

        $this->dispatcher->connect(
            'CLEP.modificacion', array('MiClaseObservador','modificacionCLEP'));
        $this->dispatcher->connect(
            'CLEP.creacion', array('MiClaseObservador','creacionCLEP'));
        $this->dispatcher->connect(
            'CLEP.consulta', array('MiClaseObservador','consultaCLEP'));
```



```
public static function creacionCLEP(sfEvent $event)
{
    //---tipo de traza---
    $tipo=new Ntrztipotraza();
    $tipo->setDenominacion($event["accion"]);

    $tipo->setActivo('TRUE');
    //---operacion de traza---
    $operacion= new Ntrzoperacion();
    $operacion->setDenominacion($event["accion"]);
    $operacion->setActivo('TRUE');
    //---subsistema de traza---
    $subsistema=new Ntrzsubsistema();
    $subsistema->setDenominacion($event["subsistema"]);
    $subsistema->setActivo('TRUE');
    //---modulo de traza---
    $modulo=new Tbrzmodulo();
    $modulo->setModulo($event["modulo"]);
    $modulo->setNtrzsubsistema($subsistema);
    //--- traza---
    $trazas= new Tbrztraza();
    $trazas->setFecha($event["fecha"]);
```

Figura 6: Ejemplo de uso del patrón Observer en el componente.

Creator (Creador): La utilización de este patrón se evidencia en la clase actions.class.php del componente, conteniendo en ella todas las funcionalidades que posee el componente.



En esta clase las acciones se encargan de crear los objetos de las clases que representan las entidades, evidenciando de este modo que la clase actions.class.php es la creadora de las entidades. (24)

```
actions.class.php x actions.class.php x listaDictamenesSuccess.php x acti
class trazasActions extends sfActions
{
    public function executePrincipal(sfWebRequest $request) {...}
    public function executeBusquedaAvanzadaTabla() {...}
    public function executeIndex(sfWebRequest $request)
    {
        $this->form = new TbttrztrazaForm();
        // sfView::SUCCESS;
    }
    public function executeNew(sfWebRequest $request)
    {...}
    public function executeCreate(sfWebRequest $request)
    {...}
    public function executeEdit(sfWebRequest $request)
    {...}
    public function executeUpdate(sfWebRequest $request)
    {...}
}
```

Figura 7: Ejemplo de uso del patrón Creator en el componente.

### 2.3.2 Tarjetas CRC (Cargo o clase, Responsabilidad y Colaboración).

Las tarjetas CRC determinan el comportamiento de cada actividad a realizar y representan un objeto. La metodología XP estipula el uso de la misma como un artefacto obligatorio durante el desarrollo de un proyecto debido a los beneficios que aportan a los desarrolladores.

El componente cuenta con las siguientes clases:

- Traza.
- Módulo.
- Subsistema.
- Tipo de traza.



- Operación.
- Excepción.
- Tipo de excepción.
- Mi Clase Observadora.

A continuación se ilustran dos de ellas escogidas por el grado de significancia de las clases que representan.

Clase: Traza

Responsabilidades	Clases relacionadas
Crear la traza con los parámetros necesarios.	Tipo de traza, Excepción, Módulo, Operación
Acceder a los diferentes parámetros de una traza.	Tipo de traza, Excepción, Módulo, Operación
Mostrar traza.	Tipo de traza, Excepción, Módulo, Operación
Seleccionar fecha como criterio de búsqueda.	Tipo de traza, Excepción, Módulo, Operación

*Tabla 6: Descripción de la CRC Traza.*

Clase: Clase Observadora

Responsabilidades	Clases relacionadas
Organizar los datos capturados y crear la traza correspondiente al mismo, así como almacenar los datos de la traza en la base de datos.	Traza, Módulo, Subsistema, Tipo de traza, Operación, Excepción, Tipo de excepción.

*Tabla 7: Descripción de la CRC Mi Clase Observadora.*

### **2.3.3 Diseño de la Base de Datos.**

Una vez realizado el diseño del componente se pasa a modelar la base de datos. El modelo de datos es una abstracción que permite visualizar las entidades de una base de datos y sus relaciones. (25), (26)



A continuación una representación del diagrama entidad relación donde se observan las entidades y sus relaciones persistentes en el modelo de datos.



Figura 8: Diagrama Entidad – Relación.

### 2.3.4 Diagrama de Despliegue

Los diagramas de despliegue son utilizados para hacer una representación de nodos, conexiones, objetos y componentes. (27)

Se ilustra el diagrama correspondiente al sistema del cual el componente de control de las trazas estará vinculado.

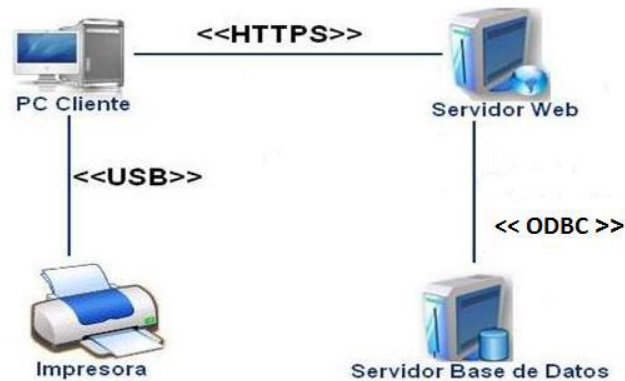


Figura 9: Diagrama de despliegue.

## 2.4 Etapa de Desarrollo

Después de terminado el diseño se procede a desarrollar la solución teniendo en cuenta los estándares de implementación, la filosofía de la programación en parejas, primordial para lograr balancear el desarrollo entre el equipo; así como las tareas de la ingeniería asociadas a las historias de usuario definidas anteriormente en la etapa de planificación; además se detallan las pruebas de aceptación a realizar sentando las bases para la validación del componente.

### 2.4.1 Estándares de codificación

El código ha de ser desarrollado siguiendo los estándares existentes para facilitar su lectura y modificación por cualquier miembro del equipo de desarrollo.

La codificación es decisiva para poder plantear con éxito la propiedad colectiva del código. Ésta sería impensable sin una codificación basada en estándares que haga más fácil para cualquier otro miembro del equipo la comprensión del código. (28)

Los estándares utilizados en la codificación fueron los siguientes:

- **Notación Húngara:** Esta convención se basa en definir prefijos para cada tipo de datos y según el ámbito de las variables. La idea de esta notación es la de dar mayor información al nombre de la variable, método o función definiendo en ella un prefijo que identifique su tipo de dato y ámbito.

**Capítulo 2: Características del Sistema**

- **Notación PascalCasing:** Es como la notación húngara pero sin prefijos. En este caso, los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula.
- **Notación CamelCasing:** Es parecido al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula.

**2.4.2 Programación en parejas**

Todo el código será desarrollado por dos personas que trabajarán de forma conjunta en un ordenador. De esta manera, se incrementa la calidad del software desarrollado sin afectar el tiempo de entrega. Se parte de la idea de que este equipo de dos personas posee unos conocimientos similares en cuanto a la tarea que van a realizar, es decir, están aproximadamente al mismo nivel. Mientras uno de ellos se encarga de pensar la táctica con la que se va a abordar el problema, el otro se encarga de pensar las estrategias que permiten llevar dichas tácticas a su máximo exponente. Ambos roles son intercambiables.

(28)

**2.4.3 Tareas de la ingeniería**

Las tareas de la ingeniería se corresponden a cada historia de usuario definida anteriormente en la planificación del componente. Se refleja a continuación las tareas asignadas a la primera historia de usuario registrar traza de autenticación.

**Tareas detalladas.**

Tarea de ingeniería	
<b>Número:</b> 1	<b>Historia de usuario (Nro.1):</b> Registrar trazas de autenticación.
<b>Nombre de la tarea:</b> Almacenar datos.	
<b>Tipo de tarea:</b> Desarrollo (Desarrollo / Corrección / Mejora)	<b>Puntos Estimados:</b> N/A.



Fecha inicio: 25/2/2012.	Fecha fin: 25/2/2012.
Programador responsable: Frank Lemus.	
<p><b>Descripción:</b></p> <p>Se accede a la aplicación. El sistema procede a almacenar los datos del usuario y la contraseña en la base de datos así como la dirección IP desde la cual se conectó.</p>	

Tabla 8: Descripción de la Tarea de Ingeniería #1.

Tarea de ingeniería	
<b>Número:</b> 2	<b>Historia de usuario (Nro.1):</b> Registrar trazas de autenticación.
<b>Nombre de la tarea:</b> Crear traza.	
<b>Tipo de tarea:</b> Desarrollo (Desarrollo / Corrección / Mejora)	<b>Puntos Estimados:</b> N/A.
Fecha inicio: 26/2/2012.	Fecha fin: 26/2/2012.
Programador responsable: Frank Lemus.	
<p><b>Descripción:</b></p> <p>Se almacena en la base de datos la traza donde se guarda dirección IP desde la cual se conectó el usuario, además de la contraseña pasando así a crear la traza con los criterios necesarios.</p>	

Tabla 9: Descripción de la Tarea de Ingeniería #2.

#### 2.4.4 Pruebas

Finalizando la etapa de desarrollo del componente quedan definidas el conjunto de pruebas a realizar en la fase final de validación de los resultados obtenidos durante el proceso de implementación del componente.

Uno de los pilares de la Programación Extrema (XP) es el proceso de pruebas. XP divide las pruebas del componente en dos grupos: pruebas unitarias, encargadas de verificar el



código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente. (29)

Las pruebas del sistema tienen como objetivo verificar que el componente cumple los requisitos establecidos por el usuario por lo que también pueden incluirse dentro de la categoría de pruebas de aceptación.

Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado al final de una iteración y el comienzo de la siguiente, por esto el cliente es la persona adecuada para diseñar las pruebas de aceptación.

#### 2.4.4.1 Pruebas Unitarias

La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y ejecutadas constantemente ante cada modificación del componente. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. (28)

#### 2.4.4.2 Pruebas de aceptación

Las pruebas de aceptación son creadas a partir de las historias de usuario. Durante una iteración la historia de usuario seleccionada se convertirá en una prueba de aceptación. Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento.

Cada una de las pruebas de aceptación representa una salida esperada del sistema. La realización de este tipo de pruebas y la publicación de los resultados debe ser lo más rápido posibles, para que los desarrolladores puedan realizar los cambios que sean necesarios. (28)

### **2.5 Conclusiones Parciales**

Con la realización de este capítulo siguiendo la metodología de software seleccionada y la utilización de las herramientas propuestas se concluyó que:

- Los requisitos definidos tributaron a que el componente cumpliera con las necesidades para las que fue diseñado.





## Capítulo 2: Características del Sistema

- Se realizó la planificación definiendo las historias de usuarios, artefacto clave para el desarrollo de la solución.
- Se implementó el componente como resultado de la investigación.
- Se conformaron los casos de pruebas necesarios para validar los resultados de la solución propuesta.



## Capítulo 3: Pruebas del Sistema.

Las pruebas de software constituyen una parte imprescindible de todo desarrollo de un sistema; una prueba no es más que una actividad en la cual un sistema o uno de sus componentes se ejecutan en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto. El objetivo de las pruebas es la detección de defectos en el software.

En este capítulo se aplican las métricas definidas para validar el diseño de la solución propuesta analizando finalmente los resultados obtenidos que permitirán comprobar la calidad del diseño. Luego se procede a realizar un conjunto de pruebas de aceptación y de unidad definidas con anterioridad para validar la solución desarrollada.

### 3.1 Métricas de Software

Las métricas son la maduración de una disciplina, que, según Pressman<sup>4</sup> van a ayudar a la evaluación de los modelos de análisis y de diseño, donde proporcionarán una indicación de la complejidad de diseños procedimentales y del código fuente, y ayudarán en el diseño de pruebas más efectivas.

Para validar el diseño se emplean dos métricas, el Tamaño Operación de Clase (TOC) y Relaciones entre Clases (RC), que ayudarán a trazar un umbral de acuerdo a varias características que se miden en cada una de ellas, las que son primordiales en todas las aplicaciones de software.

#### 3.1.1 TOC (*Tamaño Operacional de Clase*)

Esta métrica aporta valores fundamentales que permiten realizar la validación de los resultados obtenidos.

Se ilustra a continuación la aplicación de la misma obteniendo los siguientes resultados:

No	Módulo	Clase	Cantidad de Proc.	Responsabilidad	Complejidad	Reutilización
1	Traza	action.class	10	Alta	Alta	Baja
2	Traza	TbtrztrazaForm	1	Baja	Baja	Alta
3	Traza	BaseTbtrztrazaForm	2	Baja	Baja	Alta
4	Traza	TbtrztrazaPeer	8	Alta	Alta	Baja

<sup>4</sup> Roger Pressman: Ingeniería de Software un enfoque práctico, 2001.



## Capítulo 3: Pruebas del Sistema

5	Traza	TbpersonaPeer	2	Baja	Baja	Alta
6	Traza	CapturaLogs	4	Media	Media	Media
7	Traza	MiClaseObservador	15	Alta	Alta	Baja

Tabla 100: Instrumento de evaluación de la métrica TOC.

	Categoría	Criterio
Responsabilidad	Baja	$\leq 7.9$ .
	Media	Entre 7.9. y $2 * 7.9$ .
	Alta	$> 2 * 7.9$ .
Complejidad implementación	Baja	$\leq 7.9$ .
	Media	Entre 7.9. y $2 * 7.9$ .
	Alta	$> 2 * 7.9$ .
Reutilización	Baja	$> 2 * 7.9$ .
	Media	Entre 7.9. y $2 * 7.9$ .
	Alta	$\leq 7.9$ .

Tabla 111: Instrumento de evaluación de la métrica TOC.

Las figuras a continuación hacen una representación de la responsabilidad, la complejidad y la reutilización reflejando los promedios correspondientes a cada una de ellas.

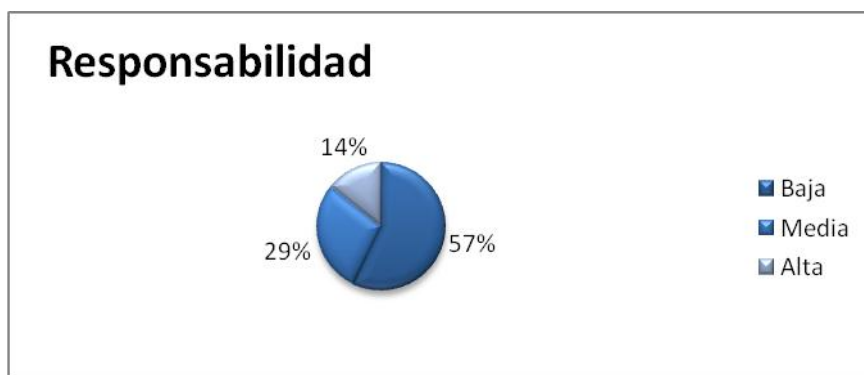


Figura 10: Resultados de la evaluación de la métrica TOC para el atributo Responsabilidad.

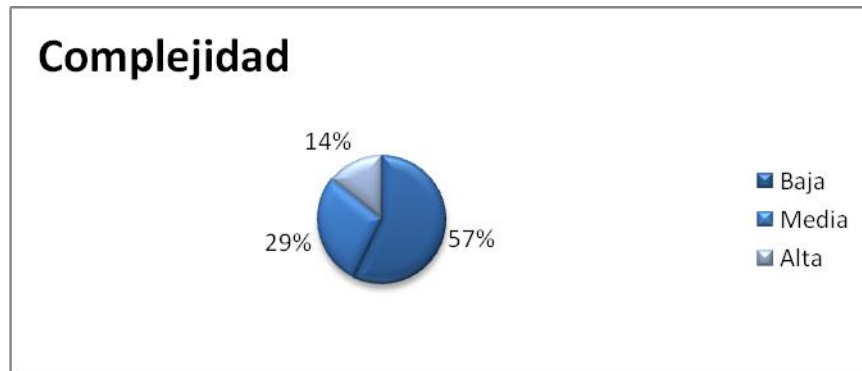


Figura 11: Resultados de la evaluación de la métrica TOC para el atributo Complejidad.

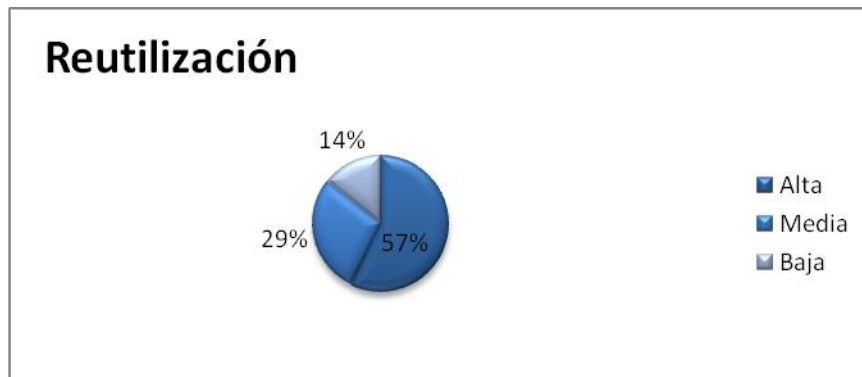


Figura 12: Resultados de la evaluación de la métrica TOC para el atributo Reutilización.

Luego de analizados los resultados obtenidos de la aplicación de la métrica TOC se puede decir que los mismos son satisfactorios y se incluyen en los rangos de calidad aceptables dentro del desarrollo de software. El 86% de las clases analizadas dieron resultados positivos en los atributos de Responsabilidad, Complejidad y Reutilización.

### 3.1.2 RC (Relaciones entre Clases)

La métrica Relaciones entre Clases (RC) está definida por el número de relaciones de uso de una clase con otra, definiendo los siguientes criterios y categorías de evaluación para los atributos de calidad:

	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1



	Medio	2
	Alto	>2
	<b>Categoría</b>	<b>Criterio</b>
Complejidad Mant.	Baja	$\leq 3$ .
	Media	Entre 3. y $2 \cdot 3$ .
	Alta	$> 2 \cdot 3$ .
	<b>Categoría</b>	<b>Criterio</b>
Reutilización	Baja	$> 2 \cdot 3$ .
	Media	Entre 3. y $2 \cdot 3$ .
	Alta	$\leq 3$ .
	<b>Categoría</b>	<b>Criterio</b>
Cantidad de Pruebas	Baja	$\leq 3$ .
	Media	Entre 3. y $2 \cdot 3$ .
	Alta	$> 2 \cdot 3$ .

Tabla 122: Instrumento de evaluación de la métrica RC.

Las figuras a continuación hacen una representación de los umbrales de acoplamiento, complejidad, reutilización y cantidad de pruebas así como el promedio reflejado para cada uno de ellos.

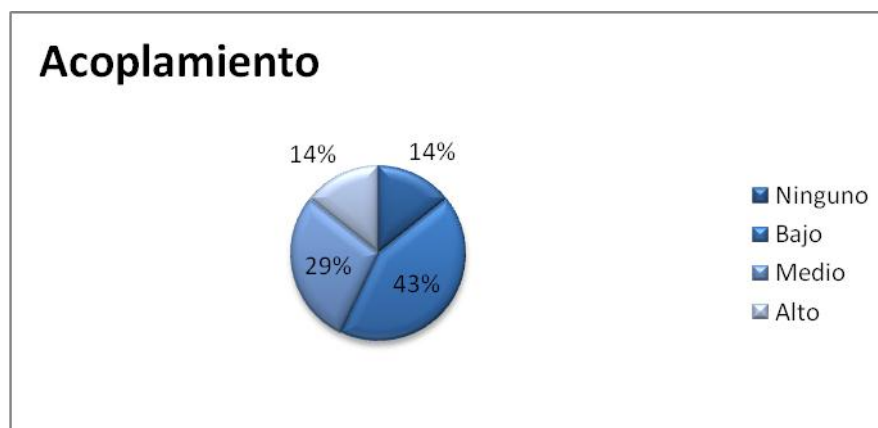


Figura 13: Resultados de la evaluación de la métrica RC para el atributo Acoplamiento.



Figura 14: Resultados de la evaluación de la métrica RC para el atributo Complejidad de Mantenimiento.

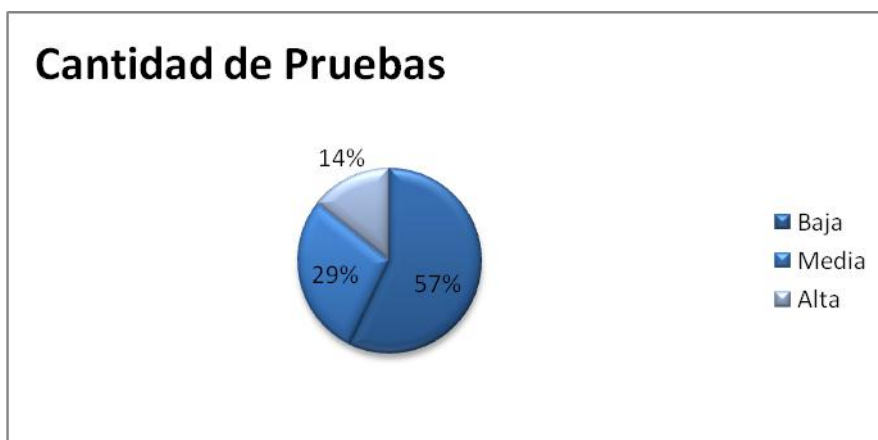


Figura 15: Resultados de la evaluación de la métrica RC para el atributo Cantidad de Pruebas.



Figura 16: Resultados de la evaluación de la métrica RC para el atributo Reutilización.



Luego de analizados los resultados obtenidos de la aplicación de la métrica RC se evidencia que el componente Gestor de Trazas tiene una calidad aceptable dentro del desarrollo de software. Para fundamentar lo antes expuesto se analizaron las clases que integran la aplicación teniendo como resultado que los porcentajes para los atributos de acoplamiento fueron bajos constituyendo una ventaja, además los datos obtenidos para los atributos de Complejidad de Mantenimiento, Reutilización y Cantidad de pruebas se comportan satisfactoriamente en el 86% de las clases.

### 3.2 Pruebas de Aceptación

Las pruebas de aceptación consisten en validar que un sistema cumple con el funcionamiento esperado, permitiendo al usuario determinar su aceptación desde el punto de vista de su funcionalidad y rendimiento.

A continuación se ejemplifican dos casos de pruebas de aceptación realizados a las historias de usuario registrar traza de acción y registrar traza de excepción, debido a la relevancia que poseen las mismas.

Caso de prueba de aceptación	
<b>Código:</b> HU3_P1	<b>Historia de Usuario:</b> 3
<b>Nombre:</b> Registrar trazas de acción.	
<b>Descripción:</b> Prueba para la funcionalidad de Registrar trazas de acción.	
<b>Condiciones de Ejecución:</b> El usuario realiza una acción en el sistema (Consulta, Modificación, Creación, etc.).	
<b>Resultados esperados:</b> El componente muestra en la interfaz la traza correspondiente a la acción realizada.	
<b>Evaluación de la prueba:</b> Prueba satisfactoria.	

Tabla 133: Descripción de Caso de prueba de aceptación: Registrar trazas de acción.



Caso de prueba de aceptación	
<b>Código:</b> HU4_P1	<b>Historia de Usuario:</b> 4
<b>Nombre:</b> Registrar trazas de excepción.	
<b>Descripción:</b> Prueba para la funcionalidad de Registrar trazas de excepción.	
<b>Condiciones de Ejecución:</b> El usuario realiza una acción que genere una excepción en el sistema.	
<b>Resultados esperados:</b> El componente muestra en la interfaz la traza correspondiente a la excepción registrada.	
<b>Evaluación de la prueba:</b> Prueba satisfactoria.	

Tabla 144: Descripción de Caso de prueba de aceptación: Registrar trazas de excepción.

### 3.2.1 Análisis de resultados

Se realizaron 8 pruebas funcionales para validar que el sistema funcionara correctamente, mostrando las salidas correspondientes a cada escenario. De estas pruebas realizadas, en una primera iteración resultaron satisfactorias 5 de ellas, representando un 62.5% del total y 3 de ellas resultaron insatisfactorias, representando un 37.5%. En una segunda iteración de un total de 8 pruebas realizadas, resultaron todas satisfactorias, constituyendo un 100% de pruebas funcionales exitosas.

## 3.3 Pruebas Unitarias

Las pruebas unitarias están dirigidas a probar la funcionalidad del código, probar las diferentes sentencias y validar que cumplan la función para las que fueron programadas proporcionando una medida de la complejidad de la implementación.

A continuación se detalla una de las pruebas unitarias desarrolladas al sistema, en este caso la del camino básico realizada a las funcionalidades `getObtenerModulo` y `executePrincipal`.

### 3.3.1 Camino Básico

El método del **camino básico** (propuesto por McCabe) permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición





de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

Para realizar estas pruebas se utiliza la **complejidad ciclomática** de McCabe, que consiste en la ejecución de un conjunto de caminos independientes proporcionando una medición cuantitativa de la complejidad lógica de un programa, así como determinar el número de casos de prueba que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

La complejidad ciclomática consta de tres formas para calcularse:

$$V(G) = A - N + 2 \text{ donde } A: \text{n}^\circ \text{ de arcos del grafo}$$

$N$ : n° de nodos

$$V(G) = R \text{ donde } R: \text{n}^\circ \text{ de regiones del grafo}$$

$$V(G) = P + 1 \text{ donde } P: \text{n}^\circ \text{ de nodos predicado}$$

A continuación se ejemplificará la aplicación la prueba de los caminos básicos de las funcionalidades `getObtenerModulo` de la clase `TbtrtrazaPeer` y el `executePrincipal` de la clase `action.class`.

### 3.3.2 Caso del método `getObtenerMódulo`:

En la siguiente imagen se muestran los posibles caminos a tomar al ocurrir la acción del método en cuestión.

```
301 static public function getObtenerModulos()
302 {
303     $traza= TbtrtrazaPeer::doSelect(new Criteria()); //1 //obtener todas las trazas para de
304     $trazaIdModulo= array(); //1
305     foreach ($traza as $trazaId) { //2
306         $trazaIdModulo[]=$trazaId->getIdmodulo(); //3
307     } //4
308     $respuesta1= array_unique($trazaIdModulo); //5 //elimina valores duplicado
309     $respuesta2= array_values($respuesta1); //5 //elimina posiciones vacias
310     $moduloArray=array(); //5 //arreglo para almacenar los
311     for ($index = 0; $index < count($respuesta2); $index++) { //6
312         $critusuario= new Criteria(); //7
313         $critusuario->add(TbtrtrazaPeer::IDMODULO,$respuesta2[$index]); //7
314         $moduloArray[]=TbtrtrazaPeer::doSelect($critusuario); //7
315     } //8
316     $respl=array(); //9
317     $respl[]='--Seleccione--'; //9
318     $trazaModulo=0; //9
319     for ($index = 0; $index < count($moduloArray); $index++) { //10
320         $trazaModulo=$moduloArray[$index][0]; //11
321         if($trazaModulo->getTbtrtrazamodulo()->getModulo()!=' ') //12
322             $respl[]=$trazaModulo->getTbtrtrazamodulo()->getModulo(); //13
323     } //14
324     $respuesta5= array_unique($respl); //15 //elimina valores duplicados
325     $respuesta6= array_values($respuesta5); //15 //elimina posiciones vacias
326     return $respuesta6; //15
}
```

Figura 17: Método `getObtenerModulo`.



Se procede a realizar los siguientes pasos:

- 1- Usando el diseño o el código como base, se dibuja el correspondiente grafo de flujo.
- 2- Se determina la complejidad ciclomática del grafo de flujo resultante,  $V(G)$ .
- 3- Se determina un conjunto básico de hasta  $V(G)$  caminos linealmente independientes.
- 4- Se preparan los casos de prueba que forzarán la ejecución de cada camino del conjunto básico.

3.3.2.1. Grafo de Flujo:

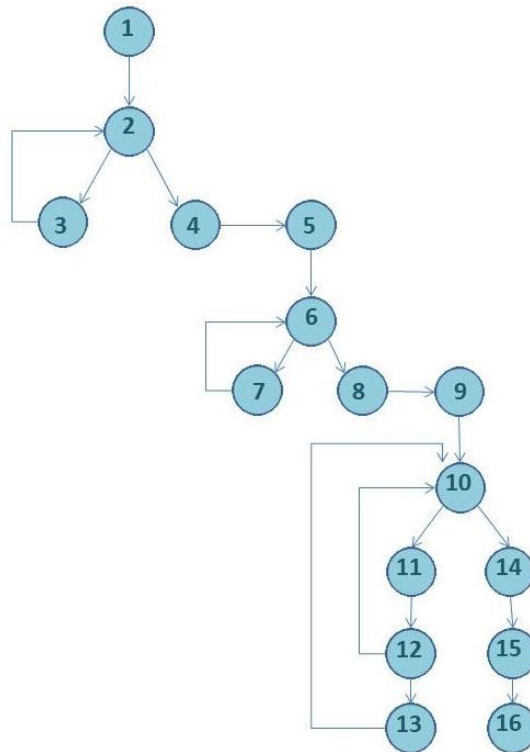


Figura 18: Grafo de Flujo del método `getObtenerModulo`.

3.3.2.2 Complejidad Ciclomática:

- 1- La complejidad ciclomática del grafo coincide con el número de regiones  $V(G)=5$ .

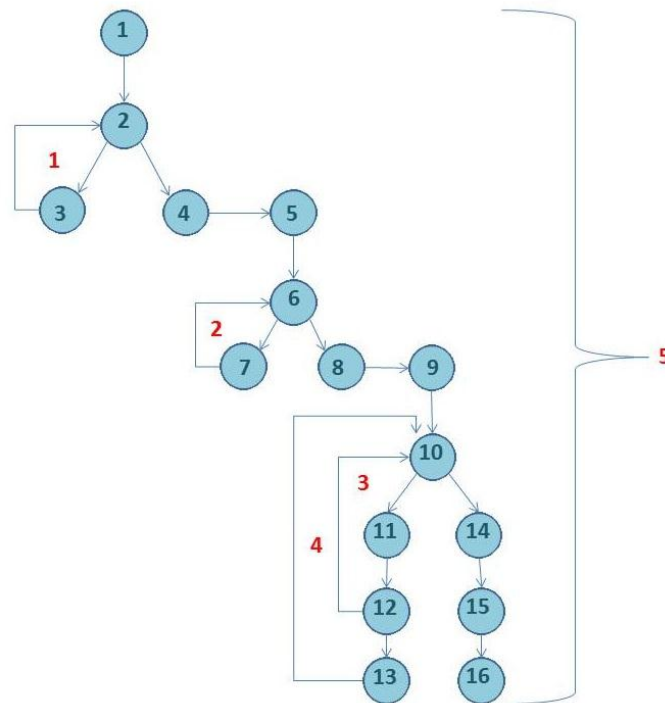


Figura 19: Grafo de Flujo del método `getObtenerModulo` dividido por regiones.

2-  $V(G) = A - N + 2 = 19 - 16 + 2 = 5$ .

3-  $V(G) = P + 1 = 4 + 1 = 5$ .

Luego de determinada la complejidad ciclomática se tiene que cinco es el número máximo de caminos a tener en cuenta para realizar las pruebas.

Caminos independientes determinados:

Camino 1: 1-2-4-5-6-8-9-10-14-15-16.

Camino 2: 1-2-3-2-4-5-6-8-9-10-14-15-16.

Camino 3: 1-2-4-5-6-7-6-8-9-10-14-15-16.

Camino 4: 1-2-4-5-6-8-9-10-11-12-10-14-15-16.

Camino 5: 1-2-4-5-6-8-9-10-11-12-13-10-14-15-16.

### 3.3.2.3 Interfaz de prueba



```
<?php
2 // test/bootstrap/propel.php
3 //include(dirname(__FILE__).'/unit.php');
4 $configuration = ProjectConfiguration::getApplicationConfiguration('PDC','dev', true);
5 new sfDatabaseManager($configuration);
6
7
8 require_once dirname(__FILE__).'/../bootstrap/unit.php';
9 $t = new lime_test(3);
10 $t->diag('This test always passes.');
```

```
Salida x
sigef2 (test:unit Trazas) x sigef2 (cache:clear) x
1..3
# This test always passes.
ok 1 - Probando metodo
ok 2 - la prueba dio correctamente
ok 3 # SKIP quitar condicional
# Looks like everything went fine.
```

Figura 20: Caso de prueba del método `getObtenerModulos`.

```
<?php
2 // test/bootstrap/propel.php
3 //include(dirname(__FILE__).'/unit.php');
4 $configuration = ProjectConfiguration::getApplicationConfiguration('PDC','dev', true);
5 new sfDatabaseManager($configuration);
6
7
8 require_once dirname(__FILE__).'/../bootstrap/unit.php';
9 $t = new lime_test(2);
10 $t->diag('This test always passes.');
```

```
Salida x
sigef2 (test:unit Trazas) x sigef2 (cache:clear) x
1..2
# This test always passes.
ok 1 - Probando metodo
ok 2 - la prueba dio correctamente
# Looks like everything went fine.
```

Figura 21: Caso de prueba del método `getObtenerModulos`.

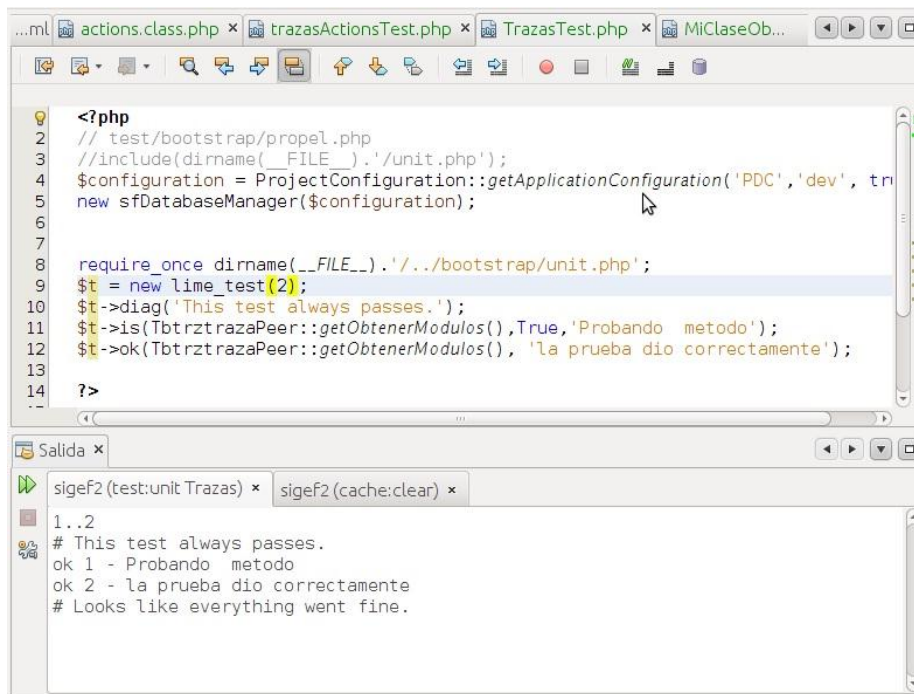


Figura 22: Caso de prueba del método `getObtenerModulos`.

### 3.3.3 Caso del método `executePrincipal`:

En la siguiente imagen se muestran los posibles caminos a tomar al ocurrir la acción del método en cuestión.

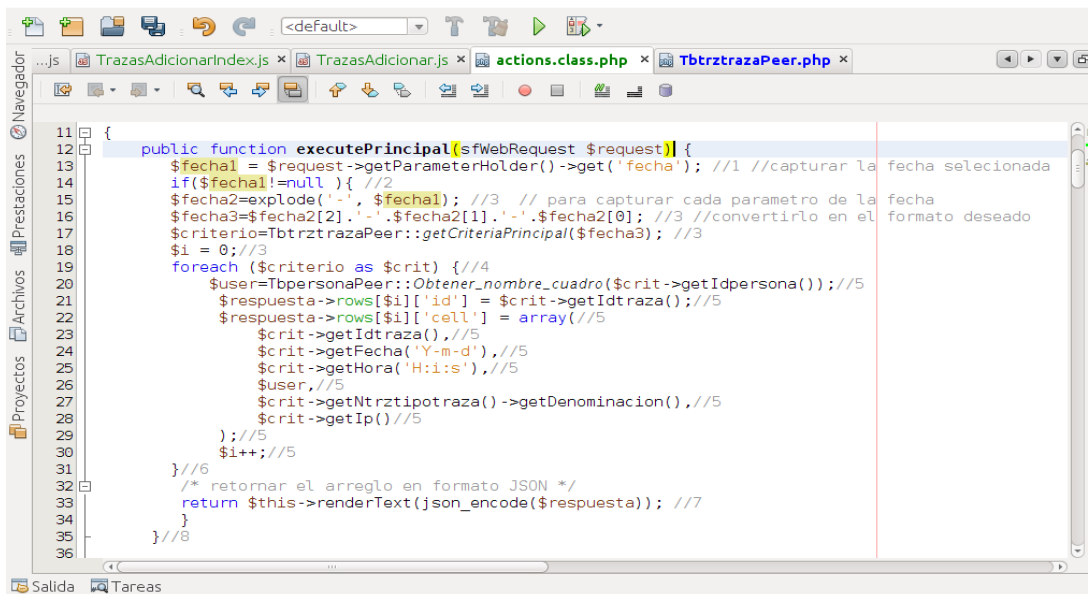


Figura 23: Método `executePrincipal`.



3.3.3.1 Grafo de Flujo:

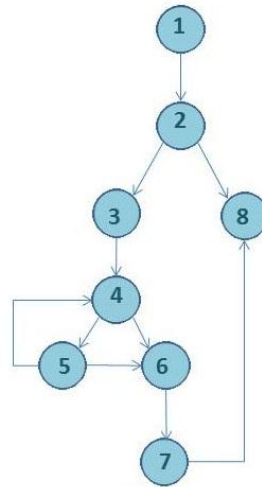


Figura 24: Grafo de Flujo del método executePrincipal.

3.3.3.2 Complejidad Ciclomática:

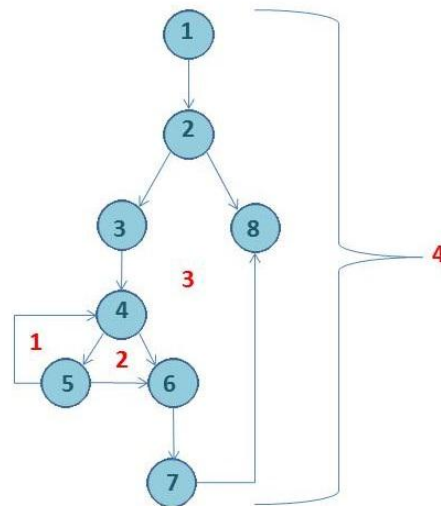


Figura 25: Grafo de Flujo del método executePrincipal dividido por regiones.

- 1-  $V(G)=4$  Coincide con el número de regiones del grafo.
- 2-  $V(G)=A-N+2=10-8+2=4$ .
- 3-  $V(G)=P+1=3+1=4$ .

Caminos independientes determinados:



Camino 1: 1-2-8.

Camino 2: 1-2-3-4-6-7-8.

Camino 3: 1-2-3-4-5-6-7-8.

Camino 4: 1-2-3-4-5-4-6-7-8.

Luego de realizados los casos de prueba para cada camino entrando valores que garanticen que se cumpla al menos una vez cada uno de estos se analizan los resultados obtenidos.

### **3.3.4 Análisis de resultados**

Para realizar la validación del código propuesto se realizaron un conjunto de pruebas a las diferentes funcionalidades del componente, en total se realizaron 20 pruebas unitarias, de las que 9 fueron aplicadas a las funcionalidades `getObtenerModulos` y `executePrincipal` respectivamente, de estas pruebas realizadas 2 de ellas resultaron insatisfactorias y en las otras 7 se obtuvieron los resultados esperados. En sentido general se detectaron 6 inconformidades, los errores detectados durante el desarrollo de estas pruebas fueron corregidos posteriormente por los desarrolladores, probadas nuevamente las funcionalidades en los caminos cuyas pruebas no cumplieron con los requisitos de aceptación en la próxima iteración del componente.

### **3.4 Conclusiones Parciales**

Luego de finalizado este capítulo se concluyeron las validaciones necesarias del componente como resultado de la investigación. Para dar cumplimiento a lo antes expuesto se aplicaron métricas que proporcionaron porcentajes satisfactorios a diferentes atributos del componente, validando la solución y permitieron comprobar su correcto funcionamiento. Además se realizaron un conjunto de pruebas a funcionalidades claves de la aplicación, obteniendo resultados que permitieron dar un valor de calidad añadido.





## Conclusiones Generales

La investigación realizada como parte del presente trabajo de diploma permite concluir que la gestión de las trazas es un requisito primordial para que las aplicaciones informáticas actuales cumplan con los estándares que permitan que las mismas sean auditables.

Teniendo en cuenta lo antes planteado con la culminación de la presente investigación se puede concluir que:

- La realización de un estudio del arte permitió sentar las bases para el desarrollo de la investigación identificando conceptos fundamentales, así como herramientas y metodologías relacionadas con la misma.
- La utilización de patrones contribuyó a un obtener un mejor diseño del componente como resultado de la investigación.
- El componente desarrollado facilita los trabajos de auditorías realizados apoyando así la detección de cualquier tipo de violación a la seguridad.
- El componente obtenido permitirá a los usuarios finales tener un control y registro de las trazas en el Sistema de Informatización de la Gestión de las Fiscalías.
- Las técnicas y métricas utilizadas para validar el componente permitieron la obtención de un entregable con un alto grado de calidad.





## **Recomendaciones**

Con vistas a mejorar el trabajo se recomienda agregar nuevas funcionalidades a la aplicación como gestionar las trazas referentes a los datos en la base de datos, así como seguir actualizando el mismo en consecuencia a la evolución del marco de trabajo y herramientas utilizadas e implementar nuevos algoritmos más eficientes en el manejo y control de las trazas.



## Bibliografía

1. Díaz Pupo, y otros, 2009.
2. Delgado Picazo, Mario. Análisis de impacto y desarrollo de buenas prácticas de auditoría en bases de datos Oracle 11G. Madrid: s.n., 2009.
3. Gonzalo Alonso Rivas, 2009.
4. Corrales Martínez, 2009.
5. Machado, Ing. Yenier Figueroa. Proyecto Técnico del Sistema de Informatización de la Gestión de las Fiscalías II. 2011.
6. Murua Olalde, Juan. Metodologías para desarrollo de software., 2009.
7. Escribano, G. F. (2002). eXtreme Programming / Programación Extrema. Recuperado de eXtreme Programming / Programación Extrema: disponible en: <http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-XP.pdf>.
8. .NET, Colectivo de Desarrollo. Arquitectura y Patrones de diseño. 2008-2009.
9. Visual Paradigm. [En línea] [Citado el: 24de 11de 2010.] <http://www.visual-paradigm.com/product/vpuml/>.
10. Sitio Oficial de Microsoft. [En línea] <http://www.asp.net/ajax>.
11. Sitio Oficial de jQuery. [En línea] <http://www.jquery.com>.
12. Sitio Oficial de Symfony [En línea] <http://www.symfony.es>.
13. PostgreSQL 8. Migración a PostgreSQL desde otras bases de datos. [en línea]. (2005). [Consultada el 10 de febrero del 2010]. Disponible en: <http://www.dbrunas.com.ar/postgres/migrapg.pdf>.
14. Netbeans [En línea] <http://www.netbeans.org>.
15. Potencier, y otros, 2009.
16. Sitio Oficial de PHP [En línea] <http://www.php.net>.
17. Eguiluz, 2009.
18. Potencier, Fabien y Zaninotto, François. Symfony, la guía definitiva. . [En línea] 2008.
19. Pressman, Roger. Ingeniería de software un enfoque práctico. 2001.
20. Pressman. Ingeniería del Software: Un Enfoque Práctico. Sexta Edición. 2005. Capítulo 15.
21. Ecured [En línea] <http://www.ecured.cu>.
22. Somerville, Ian. Ingeniería del Software, 2005.



23. Larman, Craig. UML y Patrones, Introducción al análisis y diseño orientado a objetos. México: PRENTICE HALL, 1999.
24. Teniente López Ernest, Olivé Ramón Antoni, Mayol Sarroca Enric, Gómez Seone Cristina. Diseño de Sistemas Software UML, 2003.
25. Osorio Rivera, Fray León. Base de datos relacionales.
26. Pearson Educación, Introducción a los Sistemas de Base de Datos, 2001.
27. Campderrich Falgueras, Benet. Ingeniería del Software, 2003.
28. Escribano, 2002.
29. Gutiérrez J. J., Escalona M. J., Mejías M., Mejías J. Pruebas del sistema en Programación Extrema. Sevilla, 2010.
30. Méndez Mario, 2009.
31. Ruiz-Bertol, Fran J.; Zarazaga Soria, Francisco Javier. El Control de Versiones en el aprendizaje de la Ingeniería Informática: Un enfoque práctico, 2007.
32. Eíto Brun, Ricardo. Sobre la viabilidad del código abierto. El caso de Alfresco, 2008.
33. Schwartz, Ezra. Axure Rp 6 Prototyping Essentials, 2011.
34. Otros, Jacobson y, 2000.
35. Merchán, Luis; Urrea, Alba; Rebollar, Rubén. Definición de una metodología ágil de ingeniería de requerimientos para empresas emergentes de desarrollo de software del sur-occidente colombiano, 2008. [En línea <http://www.redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=105312257003>].
36. Jeffries, R., Anderson, A., Hendrickson, C. "Extreme Programming Installed". Addison-Wesley, 2001.