**Universidad de las Ciencias Informáticas**

**Facultad 3**

**Centro de Informatización de Entidades**


# Herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe

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

**Autor(es):**

Jorge Burgos Díaz

Alicia Chacón Rodríguez

**Tutor(es):**

Ing. René R. Bauta Camejo

Ing. Katia Saria Preval

**La Habana, junio de 2015**

## 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 2015.

_____ _____

**Jorge Burgos Díaz** **Alicia Chacón Rodríguez**

_____ _____

**Ing. René Rodrigo Bauta Camejo** **Ing. Katia Saria Preval**

*"La perfección se alcanza, no cuando no hay nada más que añadir, sino cuando ya no queda nada más que quitar."*

*Antoine de Saint-Exupery*

# Agradecimientos

***De Alicia:***

*Doy gracias a Dios por mi vida y permitir todas las cosas buenas a lo largo de ella.*
*Doy gracias a Dios por mi madre por su incondicional apoyo, por ser más que un ejemplo de madre, por ser mi amiga, mi confidente, por sus largas horas en el teléfono cada vez que me derrumbaba, por todo su amor. No podía esperar más de ella.*
*Doy gracias a Dios por el gordo mío que siempre me ha cumplido todos mis caprichos, por todo su sacrificio a lo largo de mi vida, por su apoyo incondicional. A ambos muchas gracias, por brindarles este orgullo y hacerme sentir orgullosa de ser su hija. Los amo.*
*A Titi, mi flaco bello, por su paciencia, su apoyo, por soportarme todas mis malcriadeces, por ser mi compañero, por ponerme a estudiar asignaturas ya vencidas porque él no entendía, por ser mi amigo, mi confidente a lo largo de todos estos años, debí conocerte en primero. Te amo.*
*A toda mi familia en especial a mi abuela hermosa, mi abuelo por sentirse orgulloso de que una nieta suya estudie en la gran Universidad de las Ciencias Informáticas, a mis tíos Jorge y Sule por sus inmensos correos de apoyo y confianza, a todos gracias por su amor y por ser mis guardaespaldas e intercesores, muchas gracias.*
*A mi compañero de tesis por la carrera que pasamos juntos y por soportarme tanto.*
*A mi tutora por sus largas horas de dedicación y esfuerzo, por quedarse hasta altas horas de la madrugada en busca de mis malas conjugaciones, por todos sus consejos y por darme el ánimo y el apoyo para seguir luchando. Muchas gracias.*
*A todos los profesores por llenar mi cerebro de conocimientos que no daba basto, pero al final eran necesarios.*
*Al tribunal por guiarme en esta recta final con sus consejos y recomendaciones.*
*A mi segunda familia el 3503 y demás, los extrañaré a todos, A Marian por sus locuras, Araí por su "Alis, no has cogido nada nuevo", a Oda por ser la más "fashion" y mi compañera de lucha en esas largas madrugadas arreglando el documento. Anita por sus desapariciones de la UCI, a Negro por su sonrisa siempre en la cara, a Aniel por sus consejos y locuras de amigo. A Yaicel por ser el gurú de la programación. A Jose por ser mi profe cuando lo necesitaba. A Sosa por su mal genio y las grandes peleas de Firefox vs Chrome. A Tita por su maravillosa voz y Tito por sus represalias en contra de la voz de Tita, por compartir ambos momentos especiales conmigo.*
*En fin, a todos aquellos que me daban palmaditas en el hombro y decirme, claro que podemos. A todos aquellos que confiaron en mí y a los que no también porque me dieron la seguridad que necesitaba. A todos* ***Muchas Gracias****.*

# Agradecimientos

***De Jorge:***

*A mi mamá porque la palabra madre le queda pequeña.*

*A mi papá por todo su sacrificio y apoyo a lo largo de mi vida.*

*A mi hermana por ser más que una hermana, gracias por todo el amor y todo el apoyo brindado.*

***De Alicia:***

*A mis padres, por su sacrificio, amor y apoyo incondicional a lo largo de mi vida.*

***De Jorge:***

*A mis padres, y hermana por su amor y estar conmigo en todo momento*.

# Datos de contacto

**Síntesis de los tutores:**

**Tutor:** Ing. René Rodrigo Bauta Camejo.

Graduado de Ingeniería en Ciencias Informáticas en Julio del 2009. Se desempeña actualmente como Jefe del departamento de Desarrollo de Componente del Centro de Informatización de Entidades (CEIGE).

**Dirección:** Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, Km. 2 ½. Torrens, municipio La Lisa. La Habana, Cuba.

**E-mail**: rrbauta@uci.cu

**Tutora:** Ing. Katia Saria Preval.

Graduada de Ingeniería en Ciencias Informáticas en Julio 2013. Se desempeña actualmente como Analista del departamento de Desarrollo de Componente del CEIGE.

**Dirección:** Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, Km. 2 ½. Torrens, municipio La Lisa. La Habana, Cuba.

**E-mail**: ksaria@uci.cu

# Resumen

**Resumen**

La creciente informatización de los procesos empresariales en la actualidad ha propiciado la necesidad de búsquedas de herramientas que permitan su desarrollo de forma más eficiente y en el menor tiempo posible. Los marcos de trabajo son herramientas ampliamente utilizadas para el desarrollo de este tipo de sistemas pues brindan una estructura conceptual y tecnológica de soporte definida. Esta característica brinda la posibilidad a los desarrolladores de disminuir el tiempo de desarrollo y ganar en organización y calidad en el producto final. Sauxe, como marco de trabajo para el desarrollo de aplicaciones web de gestión, sirve como plataforma para el desarrollo de productos empresariales. El marco de trabajo Sauxe es utilizado en el Centro de Informatización de Entidades perteneciente a la Universidad de las Ciencias Informáticas. Actualmente la revisión del código fuente en Sauxe se hace de forma manual, lo cual es muy engorroso por el gran cúmulo de clases con que cuenta este marco de trabajo. Por la importancia que presenta el cumplimiento de la arquitectura establecida por el patrón Modelo-Vista-Controlador se hace necesario desarrollar una herramienta que permita la auditoría del código en el marco de trabajo Sauxe. Esta herramienta permitirá detectar posibles violaciones de los estándares arquitectónicos definidos por este patrón.

**Palabras claves:** auditoría, estándar, patrón, sauxe.

## Índice

# Índice de figuras

## Índice de Figuras

## Índice de Tablas

**Introducción**

En la actualidad, los sistemas de software son cada vez más importantes en la sociedad moderna, donde la calidad de los mismos es un factor clave a tener en cuenta en el proceso de desarrollo. Para lograr la calidad de un software, este debe ser sometido a pruebas para constatar que todo lo realizado esté acorde con lo pactado entre el cliente y el equipo de desarrollo.

Cuba, a partir de la década de los noventa encaminó sus esfuerzos a lograr la recuperación y desarrollo sostenible de su economía. La Universidad de Ciencias Informáticas (UCI) juega un papel primordial en los planes de desarrollo tanto económico como científico de la nación, realizando productos informáticos cada vez más eficientes y con mayor calidad. Entre sus centros de producción se encuentra el Centro de Informatización de Entidades (CEIGE), este centro cuenta con varios departamentos, entre ellos el departamento de Desarrollo de Componente el cual utiliza el marco de trabajo Sauxe, que brinda una base tecnológica para el desarrollo de aplicaciones web para la gestión empresarial.

El marco de trabajo Sauxe centra su desarrollo en los requerimientos funcionales, las interfaces de usuario y la lógica del negocio. Esta solución implementa el patrón Modelo-Vista-Controlador (MVC), donde la misma sigue los principios de soberanía tecnológica, facilidad de mantenimiento, reusabilidad e integración.

La arquitectura basada en componentes es una alternativa para que las aplicaciones desarrolladas sobre el marco de trabajo soporten el desarrollo de entidades reusables e integrables y sean de fácil mantenimiento. Cuando un desarrollador implementa alguna funcionalidad en Sauxe, debe cumplir con el esquema de organización estructural definido para este patrón arquitectónico. En este marco de trabajo han desarrollado estudiantes implementando funcionalidades que no cumplen con los estándares establecidos, lo cual viola la arquitectura definida y propicia que se debía revisar el código para evaluar el grado de correspondencia con el esquema de organización.

Actualmente la revisión de código en el marco de trabajo Sauxe se realiza de manera manual. Para la investigación sobre el proceso de revisión de código fuente, se confeccionó un cuestionario para ser respondido por los ingenieros Katia Saria Preval como analista del departamento de Desarrollo de Componente, Inoelkis Velazquez Osorio como arquitecto de datos, Mileidy Magalys Sarduy Pérez como analista principal y René Rodrigo Bauta Camejo como jefe del departamento. El cual arrojó los siguientes resultados:

- ✓ se necesitaban de 1 a 2 horas para detectar violaciones en nueve clases funcionales del marco de trabajo Sauxe.

- ✓ para llevar a cabo el proceso de reorganización de código fuente, se necesitan 29,5 horas equivalentes a tres jornadas laborales de un trabajador.
- ✓ debido a las violaciones introducidas se torna complejo el entendimiento del código y su posterior mantenimiento.
- ✓ el proceso de corregir una violación puede provocar que el sistema deje de funcionar total o parcialmente.

Ante la problemática planteada se identifica el siguiente **problema a resolver**: ¿Cómo detectar violaciones de los estándares arquitectónicos definidos para disminuir el tiempo de reorganización del código fuente así como el aumento de detección de violaciones en el desarrollo de aplicaciones web con el marco de trabajo Sauxe?

Dando respuesta al problema existente se plantea el siguiente **objetivo general:** desarrollar una herramienta para el tratamiento de violaciones de los estándares arquitectónicos definidos para el desarrollo de aplicaciones web con el marco de trabajo Sauxe para disminuir el tiempo de reorganización del código fuente así como el aumento de detección de violaciones en el desarrollo de aplicaciones web.

Del objetivo general de la investigación se derivaron los siguientes **objetivos específicos**:

1. Construir el marco teórico conceptual de la investigación sobre la auditoría de código en herramientas y plugins.
2. Identificar los requisitos funcionales y no funcionales de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe.
3. Realizar el análisis y diseño de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe.
4. Implementar la herramienta para el tratamiento de violaciones de los estándares arquitectónicos definidos del marco de trabajo Sauxe.
5. Validar la herramienta mediante pruebas funcionales.
6. Validar la investigación utilizando el método pre-experimento.

Del problema a resolver expuesto y el objetivo general planteado, se decide centrar la investigación en: las herramientas y plugins de auditoría de código como **objeto de estudio**, y como **campo de acción:** herramientas y plugins de auditoría de código para el lenguaje de programación PHP.

Como **idea a defender** se tiene que:

Si se desarrolla una herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe, ocurrirá un aumento en la detección de violaciones arquitectónicas en las aplicaciones desarrolladas con el marco de trabajo Sauxe así como la disminución del tiempo de reorganización del código fuente teniendo en cuenta los estándares definidos.

Para el correcto desarrollo del presente trabajo de diploma se emplearon **Métodos científicos** que sustentan la investigación a través de métodos teóricos y empíricos. Estos métodos servirán de guía para organizar mejor la investigación y de esta forma posibilitar el entendimiento del problema y llegar a conclusiones para la solución. A continuación se explican los seleccionados:

**Métodos teóricos:**

**Método Análisis Histórico – Lógico:** permitirá realizar un estudio de las principales soluciones relacionadas con la investigación a nivel mundial.

**Método Analítico-Sintético:** se utilizará para el estudio a partir de fuentes bibliográficas seguras de soluciones previamente desarrolladas para la auditoría de código. Este método además, permite descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.

**Métodos empíricos:**

**Entrevista:** será utilizada para obtener información acerca del proceso a investigar. A través de la entrevista se obtiene la información necesaria sobre cómo se realizan las actividades que permiten arribar a la situación problemática planteada.

El presente trabajo de diploma quedará estructurado en tres capítulos:

**Capítulo 1**: Fundamentación teórica.

En este capítulo se describen los principales conceptos relacionados con el tema. Se realizará el estudio del estado del arte que incluye un estudio de software, tecnologías y herramientas utilizadas.

**Capítulo 2:** Propuesta de solución.

El capítulo se centra en definir los temas requisitos, análisis y diseño de la solución, donde se plantean las características de la herramienta para su posterior implementación. Se expondrán los requisitos funcionales y no funcionales identificados con el que contará la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe, se muestra el modelo conceptual, modelo de datos, diagrama de clase y los resultados obtenidos en la utilización de las métricas de diseño.

**Capítulo 3:** Implementación y pruebas.

El tercer capítulo realiza una valoración de la herramienta. Expone una explicación de cómo se lleva a cabo el proceso para la implementación de la herramienta. Se validará la implementación mediante pruebas de software. Además se validará la investigación mediante el pre-experimento.

## Capítulo 1: Fundamentación Teórica

### 1.1 Introducción

A continuación se mencionan algunas de las herramientas utilizadas a nivel mundial para la auditoría de código fuente y los diferentes marcos de trabajo analizados y estudiados. Se exponen las principales características de Zend así como la extensión que surgió de él. Esta extensión junto con ExtJS que es una potente librería y Doctrine que es el Mapeador de Objeto Relacional, ayudaron a formar lo que es actualmente Sauxe. Se exponen además, algunas características y actividades de la metodología empleada en la universidad para el desarrollo de software.

### 1.2 La auditoría a estándares arquitectónicos en el código fuente

La **auditoría** es "*el proceso sistemático, que consiste en obtener y evaluar objetivamente evidencias sobre las afirmaciones relativas a los actos o eventos de carácter económico - administrativo, con el fin de determinar el grado de correspondencia entre esas afirmaciones y los criterios establecidos, para luego comunicar los resultados a las personas interesadas e independientes de conformidad con normas y procedimientos técnicos*". (Calderin, 2008) existen diferentes tipos de auditorías algunas de ellas son:

- ✓ **Auditoría contable:** se evalúa la marcha general de la empresa y se formulan recomendaciones para mejorar su gestión, la auditoría contable se limita al examen de los estados contables de síntesis, generalmente el balance y la cuenta de resultados. (Allstudies, 2015).

- ✓ **Auditoría medioambiental:** es un análisis de los efectos que sobre el medio ambiente provoca la actividad de una empresa. Además de analizar el impacto ambiental, toma en cuenta la salud y la seguridad de los trabajadores de dicha entidad. (AllPe, 2015)

- ✓ **Auditoría política:** revisión de componentes de campaña que permite realizar ajustes o validar decisiones, hacer las modificaciones pertinentes y realizar una valoración de las decisiones, con la finalidad de mejorar los esfuerzos para llegar a los electores. (Krathos, 2015)

- ✓ **Auditoría informática:** conjunto de procedimientos y técnicas para evaluar y controlar total o parcialmente un sistema informático, con el fin de proteger sus activos y recursos, verificar si sus actividades se desarrollan eficientemente y de acuerdo con la normativa informática y general existente en cada empresa y para conseguir la eficacia exigida en el marco de la organización correspondiente. (Castro, 2012)

Las entidades que utilizan aplicaciones web para la gestión de sus procesos hacen uso de la auditoría o revisión de código fuente con el objetivo de analizar su aplicación.

Según el gerente de desarrollo de la empresa Caraytech S.A e integrante del prestigioso grupo Pragma consultores, Gabriel Naiman **la auditoría o revisión de código** consiste en el análisis estático del código fuente de una aplicación, esto significa que no se revisa el flujo de ejecución de los programas, sino la adherencia a estándares o buenas prácticas utilizados en la codificación. (Naiman, 2009)

Para llevar a cabo una buena seguridad, control y organización se hace necesario que el código fuente sea lo más organizado posible. En informática se conoce como **código fuente** a todas las series de instrucciones y estilos o estándares que puede utilizar un desarrollador de software para construir un producto o programa, donde el ordenador debe ser capaz a través de un compilador ejecutar dichas sentencias. (Leahy, 2015).

Según Evelio Martínez en (Martínez E., 2014) define como **estándar** a "*normas documentadas que contienen especificaciones técnicas u otros criterios precisos para ser usados consistentemente como reglas, guías, o definiciones de características. Para asegurar que los materiales, productos, procesos y servicios se ajusten a su propósito*".

Según Humberto Cervantes en (Cervantes, 2010) la **arquitectura de software** se refiere a la estructuración del sistema donde se representa un diseño de alto nivel del sistema que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad), y servir como guía en el desarrollo.

Se define **arquitectónico** como las cualidades, características de la arquitectura. (Dictionary, 2015).

Teniendo en cuenta las definiciones anteriores, se entiende como **estándares arquitectónicos** al conjunto de reglas, normas y características que permitan asegurar el diseño y la implementación de un software.

En el marco de la investigación y teniendo en cuenta los conceptos citados anteriormente, la **auditoría a estándares arquitectónicos** se define como el proceso de evaluación y control al conjunto de reglas definidas en la estructuración de un sistema informático.

Según Gabriel Naiman el **proceso de revisión** se basa en las definiciones de un conjunto de reglas, las cuales se verifican sobre el código a revisar, para finalmente obtener una lista de errores.

### 1.3 Herramientas utilizadas a nivel mundial para auditar código fuente

Para que una aplicación o producto de software pueda ser usado debe transitar por una gestión de revisión y pruebas. Para ello existen los **analizadores de código** que no son más que **herramientas** que realizan la lectura del código fuente y devuelven observaciones o puntos en los

que el código puede mejorarse desde la percepción de buenas prácticas de programación y código limpio.

Según Javier Garzás profesor titular de la universidad de referencia Rey Juan Carlos de Madrid, doctor y experto en gestión de proyectos y equipos, dentro del amplio mundo de las herramientas de calidad de software las más populares y recomendadas son las descritas a continuación. (Garzás, 2012)

- ✓ **PMD**

Analizador estático de código que utiliza unos conjuntos de reglas para identificar problemas dentro del software. Detecta elementos como código duplicado, código muerto (variables, parámetros o métodos sin usar), complejidad de métodos, malas prácticas de programación. Trabaja principalmente con lenguaje Java, aunque, con menos soporte, también posee conjuntos de reglas para JavaScript, XSL. (Garzás, 2012)

Dentro de sus características se destacan: (Gutiérrez & Escalona, 2008)

- ✓ Integrable en entornos como Netbeans o Eclipse y es compatible con otros similares.
- ✓ Detecta código poco óptimo, expresiones redundantes, sentencias *if* innecesarias.
- ✓ Detecta código que no se usa: variables locales, parámetros y métodos privados.
- ✓ Trae implementada un conjunto de reglas por defecto priorizadas de 5 (leve) a 1 (muy crítica).

Esta herramienta tiene la ventaja que es integrable con el Netbeans ya que viene empaquetado como una herramienta más del mismo que ayudará a obtener un código más elegante, sencillo y fácil de mantener.

- ✓ **CheckStyle**

Es una herramienta de desarrollo para ayudar a los programadores a escribir código Java que se adhiere a un estándar de codificación. Automatiza el proceso de comprobación de código Java. Presenta la ventaja de ser altamente configurable. También tiene capacidad para comprobar problemas de diseño de código y de formato. (Checkstyle, 2001)

Según Alexis Vilañez en (Vilañez, 2014) el conjunto de reglas disponible es muy completo y está clasificado en:

- ✓ Comentarios Javadoc: facilitar el mantenimiento, permite obligar a comentar los nombres de clases, todos los métodos menos get/set y los atributos públicos.
- ✓ Convenciones de nombres: posibilita definir una expresión regular para el nombre de todo.
- ✓ Cabeceras: expresiones regulares para las cabeceras de los ficheros.
- ✓ Imports: reglas para los import, como no usar *, entre otros.
- ✓ Violaciones de tamaño: define un máximo para el tamaño de las clases, métodos, líneas y número de parámetros de un método.

- ✓ Espacios en blanco: reglas para definir dónde se ponen espacios en blanco y tabuladores en el código.

CheckStyle se basa fundamentalmente en un estilo de código para realizar un producto. Tener bien definido un buen estilo de programación puede ayudar a cumplir con las buenas prácticas que mejoran la calidad del software, la legibilidad, la reutilización y reducir el costo del desarrollo. Sin embargo, los controles efectuados limitan en la presentación y no analizan el contenido. En la práctica, puede ser complejo cumplir con todas las restricciones de estilo, algunas pueden ser perjudiciales para la fase de programación dinámica.

✓ **SONAR**

Es una herramienta de software libre y gratuito que permite gestionar la calidad del código fuente. La misma cubre siete ejes: arquitectura y diseño, comentarios, normas de codificación, errores potenciales, complejidad, duplicaciones y pruebas unitarias. (SonarQube, 2008)

Posee varias características destacando: (SonarQube, 2008)

- ✓ Cubre varios idiomas, añadiendo motores de reglas.
- ✓ Permite descubrir rápidamente los proyectos y/o componentes que están en deuda técnica para establecer planes de acción.
- ✓ Es una aplicación web donde las reglas, alertas, exclusiones y ajustes se pueden configurar en línea.
- ✓ Posee más de 20 lenguajes de programación entre ellos PHP, pero trabaja principalmente para Java.

Sonar posee una amplia librería que puede recopilar, analizar y visualizar métricas del código fuente. Sin embargo lo que se analiza variará en función del idioma.

✓ **Google CodePro Analytix**

Actualmente esta herramienta de prueba de software es muy importante para los desarrolladores del IDE[1] de programación Eclipse ya que la misma mejora la calidad y los costos del software. Esta herramienta ayuda a mantener buenas prácticas de codificación en línea con las directrices de la organización.

CodePro Analytix es una herramienta dinámica gratuita que trabaja formando parte como plugins de Eclipse, la misma permite detectar, informar y reparar desviaciones o incumplimientos de las normas predefinidas de codificación, se enmarca en segmentos duplicados o muy similares (producto del copiar y pegar).

Ofrece un entorno para evaluación de código, métricas, análisis de dependencias, cobertura de código y generación de test unitarios, entre otros.

[1] **IDE:** por sus siglas en inglés de **I**ntegrated **D**evelopment **E**nvironment.

Esta herramienta realiza una transformación en el código preservando su comportamiento, modificando solo su estructura interna para mejorarlo. El objetivo principal es disminuir la complejidad del software en anticipación a los incrementos de complejidad que los cambios pudieran traer consigo. (Developers, 2012)

✓ **Simian**

Es una herramienta gratuita, detecta código duplicado que es el mayor enemigo de la mantenibilidad en desarrollos realizados con los lenguajes: Java, C#, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML y Visual Basic.

Evita problemas como: (Garzás, 2011)

- ✓ Aumenta innecesariamente el número de líneas de código, y está comprobado que a más líneas de código (no necesarias) más complejo es el mantenimiento.
- ✓ El proceso de buscar repeticiones se torna complejo ya que se debe de buscar en varios sitios donde se escribe, resultando un software incoherente. Por ello baja la productividad y aumenta la probabilidad de error.
- ✓ Normalmente un problema de duplicación de código esconde un problema de diseño software.

### 1.3.1 Comparación entre las herramientas estudiadas

**Tabla 1. Comparación de herramientas.**

| | PMD | CheckStyle | Sonar | CodePro Analytix | Simian |
|---|---|---|---|---|---|
| **Tecnologías Libres** | Sí | Sí | Sí | Sí | Sí |
| **Usabilidad del sistema** | Sí | Sí | Sí | Sí | Sí |
| **Lenguaje de programación** | Java | Java | Java, PHP entre otros | Java | Java, C#,C,C++,HTML entre otros |
| **Integración con marcos de trabajo** | No | No | No | No | No |

### 1.3.2 Resultados del análisis

Las herramientas para la revisión de código descritas en la investigación centran sus objetivos y especificaciones en detectar errores cometidos en el proceso de desarrollo. Con respecto a los

indicadores a tener en cuenta, fueron escogidos de acuerdo a los requerimientos que posee el marco de trabajo Sauxe, donde se puede decir que todas las herramientas estudiadas proveen un entorno de fácil uso mediante el cual se obtienen resultados de forma sencilla, lo que provoca un factor importante de usabilidad. Por otra parte, las herramientas estudiadas son libres y gratuitas lo que las hace de gran aceptación y beneficio a todo aquel programador que desee revisar su código. Mediante este estudio se pudo constatar que la gran mayoría de estas herramientas son instaladas en IDEs de programación fundamentalmente Eclipse y Netbeans y son empaquetadas como una herramienta más de los mismos.

A diferencia de Sonar y Simian, las herramientas CheckStyle, PMD y CodePro Analytix se basan en evaluaciones de código fuente para el lenguaje de programación Java. Sonar posee una gran librería que da soporte a múltiples lenguajes incluyendo a PHP. Otro punto importante es que la auditoría que realizan se basa en errores de inconsistencias de variables, código duplicado, establecimiento de un estilo de código entre otros.

A pesar de no ser posible utilizarlas para los propósitos del marco de trabajo, sí se considera que aportan contenido teórico así como buenas prácticas como:

- ✓ evitar el uso de variables inconsistentes en métodos que no le corresponden, como en los marcos de trabajo el código es reutilizable, se debe de tener en cuenta la copia y pega de código provocando que el posterior mantenimiento sea engorroso.
- ✓ evitar la creación innecesaria de objetos.
- ✓ importar las clases específicas para la realización de las aplicaciones.
- ✓ evitar utilizar clases o métodos obsoletos.

Todo esto contribuirá a que la implementación tenga bases sólidas y que el código esté lo más organizado posible. Reafirmando así la necesidad de crear una solución informática que se adecue a la problemática existente.

## 1.4 Metodología de desarrollo

Para el desarrollo de cualquier producto de software se realizan una serie de tareas entre la idea inicial y el producto final.

Se define por **metodología** "*un conjunto de procedimientos, técnicas y ayudas a la documentación para estructurar, planificar y controlar el proceso de desarrollo de un sistema de información."* (Knowledge, 2015)

En la Universidad de las Ciencias Informáticas se utiliza la Metodología de desarrollo para la Actividad productiva de la UCI. En la que se detalla el ciclo de vida de sus proyectos y tiene como objetivo aumentar la calidad del software. Esta metodología establece distintas fases por la que debe transitar un producto durante el desarrollo, y el conjunto de productos de trabajo que se generan en cada una de ellas. Las fases definidas son: *Inicio, Ejecución y Cierre.* (Sánchez, 2014)

### 1.4.1 Flujo de trabajo

En el marco de la siguiente investigación el flujo de trabajo comprende las disciplinas pertenecientes a la fase de Ejecución, las que se muestran en la siguiente figura:

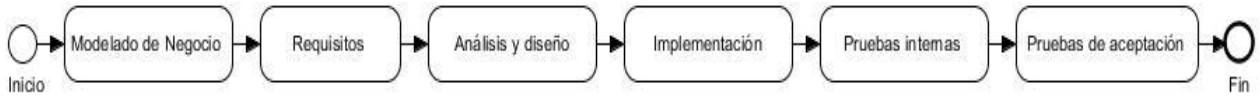



**Figura 1. Flujo de datos.**

Durante el flujo de trabajo en la disciplina de **Modelado de negocio** se emplea el modelo de dominio, no se utiliza modelo de procesos debido a que para la herramienta que se desea desarrollar no existen procesos de negocio definidos. En esta disciplina, para el modelo de dominio se genera el modelo conceptual, el cual describe los aspectos del dominio, identificándose los principales elementos físicos o lógicos que ayuden a entender el problema y que generalmente se presentan como clases. El modelo conceptual explica cuáles son y cómo se relacionan los conceptos relevantes en la descripción del dominio de un problema, identificando atributos y asociaciones existentes entre ellos.

En la disciplina **Requisitos** se identifican los requisitos funcionales, luego se describen, se elaboran los prototipos de interfaces de usuario asociados a ellos y posteriormente se validan. Además se identifican también los requisitos no funcionales. Durante la disciplina de **Análisis y diseño** se modela la herramienta, definiéndose el patrón arquitectónico que determina la estructura fundamental de la herramienta. Se genera el modelo de diseño, artefacto conformado por los diagramas de clases del diseño con estereotipos web, mientras que para finalizar esta disciplina se realiza el modelo de datos y se elaboran los diseños de casos de prueba, así como la validación del diseño realizado. En la disciplina **Implementación** se elabora el diagrama de componentes y se implementan los requisitos funcionales identificados con sus correspondientes interfaces. En la disciplina **Pruebas internas** se realizan pruebas de caja negra mediante la técnica de partición equivalente y las pruebas de caja blanca utilizando la técnica del camino básico, también se resuelven las no conformidades detectadas durante la ejecución de estas pruebas. Finalmente se encuentran las **Pruebas de aceptación** donde se verifica que el software esté listo y que puede ser usado por usuarios finales. (Sánchez, 2014)

## 1.5 Arquitectura de software

La arquitectura de software juega un papel esencial en la descripción de alto nivel de todas las partes que conforman el sistema. (IEEE, 2012).

La mayoría de los autores coinciden en que una arquitectura de software define la estructura del sistema. (Camacho, 2004). En la arquitectura de software se determina el estilo arquitectónico y el patrón arquitectónico, donde:

Según Buschmann se define como **estilo arquitectónico** a una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación, así como las reglas para su construcción. (Camacho, 2004). Los principales estilos arquitectónicos, así como una pequeña descripción de ellos son los que se muestran en la Tabla 2.

**Tabla 2. Estilos arquitectónicos.**
**Fuente: (Erika Camacho, 2004)**

| Estilo | Descripción |
|---|---|
| **Datos Centralizados** | Sistemas en los cuales cierto número de clientes accede y actualiza datos compartidos de un repositorio de manera frecuente. |
| **Flujo de Datos** | El sistema es visto como una serie de transformaciones sobre piezas sucesivas de datos de entrada. El dato ingresa en el sistema, y fluye entre los componentes, de uno en uno, hasta que se le asigne un destino. |
| **Máquinas Virtuales** | Simulan alguna funcionalidad que no es nativa al hardware o software sobre el que está implementado. |
| **Llamadas y Retorno** | El sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas. |
| **Componentes Independientes** | Consiste en un número de procesos u objetos independientes que se comunican a través de mensajes. |

Para un mejor entendimiento de patrón arquitectónico se hace necesario conocer que es un **patrón** el cual Buschmann plantea que es una solución probada que se puede aplicar con éxito a un determinado tipo de problema que aparece con frecuencia.

Partiendo del concepto anterior se puede definir que un **patrón arquitectónico** según Buschmann "*expresa esquemas de organización estructural fundamentales para los sistemas de software. Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen guías y lineamientos para organizar las relaciones entre ellos*." (Buschmann, 1996). La Tabla 3 muestra algunos patrones arquitectónicos así como una breve descripción de acuerdo a Buschmann.

**Tabla 3. Patrones arquitectónicos.**
**Fuente: (Erika Camacho, 2004)**

| Patrón | Descripción |
|---|---|
| **Capas** | Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de abstracción. |

| | |
|---|---|
| **Tubería y Filtro** | Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro. El dato pasa a través de conexiones, entre filtros adyacentes. |
| **Pizarrón** | Aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial o aproximada. |
| **Modelo-Vista-Controlador** | Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controlers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario. |
| **Reflexión** | Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones. |

## 1.6 Herramientas y tecnologías

El desarrollo exitoso de los sistemas informáticos, depende en gran medida de una correcta elección de las herramientas y tecnologías a utilizar. Las definidas en la Vista Entorno de Desarrollo Tecnológico de Sauxe, se describen brevemente a continuación:

**Herramienta para el modelado**

- ✓ **Visual Paradigm 8.0** es una herramienta UML[2] profesional que soporta el ciclo de vida completo del desarrollo de software: requisitos, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste. (Paradigm, 2013)

**Herramientas de base de datos**

- ✓ **PostgreSQL 9.0** es un sistema de gestión de bases de datos. Utiliza un modelo cliente/servidor que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. (Martínez, 2013)

[2] **UML**: por sus siglas en inglés de Unified Modeling Language.

- ✓ **PgAdminIII** es una interfaz de administración para gestionar bases de datos PostgreSQL. Permite desde ejecución de consultas SQL simples, hasta la elaboración de bases de datos complejas. (PgAdmin, 2011)

**Herramientas para el control de versiones**

- ✓ **Subversion** es un sistema de control de versiones muy utilizado para el mantenimiento de código fuente, documentación técnica y páginas web. (Fabri, 2015)
- ✓ **RapidSVN 1.6.6** es un cliente gráfico para Subversion. Es fácil de usar, tanto para los que conocen Subversion como para los que empiezan, pudiendo acceder a direcciones SVN, subir y descargar contenido y sincronizarlo con el servidor original, comprobar su estado, crear y fusionar direcciones. (López, 2010)

**Herramientas para el desarrollo**

- ✓ El **servidor web Apache v2.0** es una tecnología gratuita, de código abierto y altamente configurable de diseño modular por lo que resulta muy sencillo ampliar sus capacidades. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor y es posible configurarlo para que ejecute un determinado script cuando esto suceda. (Apache, 2014).
- ✓ El IDE **Netbeans IDE 8.0** es un reconocido entorno de desarrollo integrado de código abierto y una plataforma de aplicación que permite a los desarrolladores crear con rapidez aplicaciones web, empresariales, de escritorio y móviles utilizando la plataforma Java, así como JavaFX, PHP, JavaScript y Ajax. (Netbeans, 2014)

## 1.7 Framework de desarrollo

Se define por **framework** al conjunto de componentes que forman un diseño reutilizable que facilita y agiliza el desarrollo de sistemas web. Los objetivos principales que persiguen son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. (Gutiérrez, 2009).

Son desarrollados con el objetivo de brindar a los programadores y diseñadores una mejor organización y estructura a sus proyectos.

**Zend Framework 1.11** está desarrollado por Zend bajo el criterio OpenSource para PHP 5.0+. Implementa el patrón Modelo-Vista-Controlador y es completamente orientado a objetos. Esta arquitectura es de acoplamiento flexible lo cual permite a los desarrolladores utilizar cualquier componente. (Framework, 2015)

**Doctrine 1.2.2** es un mapeador de objeto relacional (ORM[3]) para PHP 5.0+ que se encuentra en la parte superior de la capa de abstracción de base de datos. Permite escribir las consultas de base de datos en un dialecto orientado a objetos de propiedad SQL, que mantiene la flexibilidad sin necesidad de duplicar código innecesario. (Doctrine, 2015)

**ExtJS v4** es una librería JavaScript de código abierto de alto rendimiento para la creación y desarrollo de aplicaciones web dinámicas. Provee interfaces gráficas de usuario que brindan experiencias parecidas o iguales a las que se tienen con aplicaciones de escritorio. Permite la creación de aplicaciones complejas utilizando componentes predefinidos. Entre sus principales ventajas se encuentra el balance entre Cliente-Servidor, distribuyendo la carga de procesamiento en el último, y este al tener menor carga, maneja los clientes de manera más eficiente. La comunicación asíncrona permite el intercambio de información con el servidor sin necesidad de pedirle una acción al usuario, dando la libertad de cargar la información sin que lo note. (Sencha, 2014)

**El marco de trabajo Sauxe** es un marco de trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. (Baryolo, 2010)

## 1.8 Lenguaje de modelado y desarrollo

### 1.8.1 Lenguaje para el modelado

El **Lenguaje Unificado de Modelado (UML)** es un lenguaje de modelado estandarizado de propósito general en el campo de la ingeniería de software orientada a objetos. UML incluye un conjunto de técnicas de notación gráfica para crear modelos visuales de programación orientada a objetos. (Pressman, 2005)

UML proporciona una forma estándar de escribir planos de sistemas, que abarcan clases escritas en un lenguaje de programación específico, esquemas de bases de datos y componentes de software reutilizables. (Academia, 2015)

UML representa para los desarrolladores de aplicaciones y sistemas una serie de ventajas, al igual que para las organizaciones, entre estos beneficios destacan: (Kusek, 2001)

- ✓ Aumento notable de la capacidad de trabajo y de diseño a los desarrolladores que lo utilizan.
- ✓ Mejora el proceso de diseño de software.
- ✓ Fortalecimiento del proceso de diseño de la estructura dinámica de los sistemas.

[3] **ORM**: por las siglas en inglés de **O**bject **R**elation **M**apper.

### 1.8.2 Lenguaje de programación

Un **lenguaje de programación** es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo. (Kioskea, 2015)

**PHP[4] 5** es un lenguaje de código abierto interpretado, de alto nivel, está orientado al desarrollo de aplicaciones web, que son interpretadas del lado del servidor. Su rapidez en la ejecución y los bajos requerimientos de consumo en los sistemas donde es desplegado lo hacen uno de los preferidos por los desarrolladores.

Su mayor ventaja radica en ser un lenguaje libre, por lo que se convierte en una alternativa de muy fácil acceso, además de poseer una comunidad de desarrolladores que intercambian experiencias, de esta forma cuando se presenta un problema, es muy fácil obtener documentación para darle solución de forma rápida y sin costo alguno. (Mehdi Achour, 2014)

**JavaScript** es un lenguaje basado en objetos, que se utiliza principalmente para crear páginas web dinámicas y permite el desarrollo de interfaces de usuario mejoradas. Una página web dinámica es aquella que permite la interacción entre el contenido de la misma y el usuario. JavaScript permite incorporar a dichas páginas efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (JavaScript, 2008)

### Conclusiones del capítulo

La revisión bibliográfica permitió conocer y caracterizar los conceptos fundamentales relacionados con la situación problemática que contribuyen a un mejor entendimiento del contexto.

El análisis de varias soluciones informáticas para la auditoría de código arrojó como resultado que teniendo en cuenta sus características, a pesar de no utilizar ninguna de las referidas, su estudio sí aporta buena base para la elaboración de la herramienta.

El análisis de la Metodología de desarrollo para la Actividad productiva de la UCI, promovió la comprensión de sus principales características y sus fases en pos de guiar el posterior diseño de la solución. De igual forma, el estudio de las herramientas y tecnologías definidas por el departamento para el desarrollo de sus productos, garantiza un buen punto de partida en la

[4] **PHP**: por sus siglas en inglés Hypertext Preprocessor

construcción de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe.

# Capítulo 2: Propuesta de solución

## 2.1 Introducción

En el presente capítulo se realiza una propuesta de solución de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe. Se describen los requisitos funcionales y no funcionales con que contará la herramienta. Se muestran historias de usuario de algunos de los requisitos funcionales. Se describe el patrón arquitectónico utilizado, así como los patrones de diseño. Se muestran los diferentes artefactos generados durante las disciplinas requisitos y análisis y diseño.

## 2.2 Propuesta de solución

Según Gabriel Naiman el proceso de revisión consta de cuatro etapas: planificación, definición de reglas, ejecución y reporte de resultados. En la etapa de ejecución se ponen en práctica los controles y los estándares definidos en la segunda etapa, para realizar este proceso se podrían utilizar herramientas para realizar el proceso de revisión de forma automática. (Naiman, 2009)

Para ello, en el marco de trabajo Sauxe con la herramienta se pretende mejorar la detección de violaciones arquitectónicas de las aplicaciones desarrolladas con el marco de trabajo Sauxe así como disminuir el tiempo de reorganización del código fuente teniendo en cuenta los estándares definidos.

La herramienta formará parte integrante del mismo, posibilitando cargar ficheros de clases ya sean controladoras, modelos o clases de datos, revisando si contienen violaciones en los estándares y generando un informe detallado de las violaciones encontradas y sugerencias de como mitigarlas.

En la solución se analizarán las siguientes capas:

- ✓ **Capa de acceso a datos:** en esta capa se encuentran la clase *domain* en la cual no se pueden consumir ningún tipo de servicio. Además no se pueden utilizar variables globales. Ejemplos (*Global_Concept, Zend_Registry, $SESSION*).
- ✓ **Capa de negocio:** en esta capa se encuentran las clases controladoras y las *model*. En las primeras no se puede tener llamadas a la capa de acceso a datos como tampoco se puede tener llamadas a ningún servicio. En las clases *model* no se puede realizar llamadas a la capa de acceso a datos de otra funcionalidad y realizar llamadas de *model* a *model*.

Después de revisar el código de estas clases se deben visualizar las violaciones detectadas en las clases agrupadas por tipo de fichero (*controller*, *model*, *domain*), los tipos de violaciones, la sentencia de código así como el número de línea donde se encuentre ésta. Una vez detectadas las violaciones se debe sugerir la clase donde debe ir esa sentencia para corregir esa violación.

## 2.3 Modelo conceptual

El modelo conceptual podría considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio. Mediante un lenguaje visual se pueden representar las conceptos del mundo real que son de interés para el dominio de forma más sencilla y entendible. (Larman, 1999).

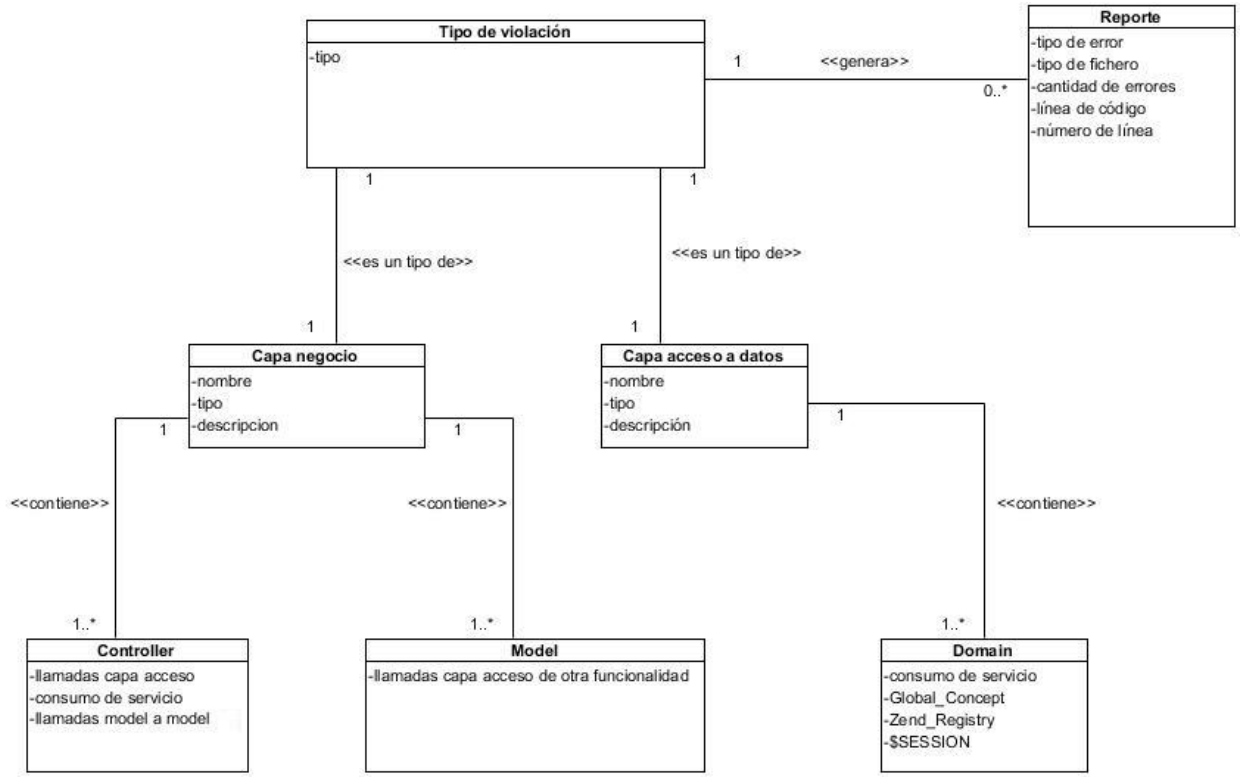



**Figura 2. Modelo conceptual.**

A continuación se describen los principales conceptos representados en el modelo conceptual perteneciente a la herramienta:

- ✓ **Violación de estándares arquitectónicos:** agrupa a los tipos de violaciones que se encuentran en las capas de acceso a datos y negocio.
- ✓ **Capa de negocio:** agrupa a los diferentes tipos de ficheros posibles en esa capa, en la misma se encuentran las clases controladoras y las modelos.
- ✓ **Controller:** agrupa los diferentes tipos de violaciones que puede contener la clase controladora.
- ✓ **Model:** agrupa los diferentes tipos de violaciones que puede contener la clase modelo.
- ✓ **Capa de acceso a datos:** agrupa a los diferentes tipos de ficheros posibles en esa capa, en ella se encuentra las clases *domain* las cuales se encargan de las consultas a la base de datos.
- ✓ **Domain:** agrupa los diferentes tipos de violaciones que contiene la clase *domain*.
- ✓ **Reporte:** agrupa toda la información referente a las violaciones encontradas.

## 2.4 Requisitos del software

El equipo de desarrollo debe determinar las técnicas que se deberán usar según el producto a desarrollar para la captura de requisitos. Una opción factible para realizar este proceso es combinar varias de estas técnicas. Las utilizadas para la captura de los requisitos de la herramienta fueron: tormenta de ideas y entrevista.

Los implicados fueron:

- ✓ Analista del departamento de Desarrollo de Componente la Ing. Katia Saria Preval.
- ✓ Analista principal del departamento de Desarrollo de Componente la Ing. Mileidy M. Sarduy Pérez.
- ✓ Arquitecto de datos el Ing. Inoelkis Velazquez Osorio.
- ✓ Los estudiantes analista y desarrollador involucrados en el desarrollo de la herramienta.

Después de amplios debates donde todos los implicados mencionados anteriormente expusieron sus puntos de vistas, sin prejuicios y valoraciones que pudieran descartar las ideas del resto de los participantes, se logró una aproximación inicial a los requisitos que se implementarían en la herramienta.

Después de construir los primeros prototipos de la herramienta, fueron presentados al cliente, el cual refinó los requisitos que se reflejaban en los mismos y adicionó nuevos requisitos que no se habían logrado detectar durante la tormenta de ideas y la entrevista.

### 2.4.1 Requisitos funcionales

Durante la fase de captura de requisitos se identificaron 10 requisitos funcionales de los cuales ocho están incluidos en tres agrupaciones como se muestra en la siguiente tabla. Con el propósito de lograr un mayor entendimiento de los mismos, se realizaron las historias de usuario de cada uno de ellos.

**Tabla 4. Listado de requisitos funcionales.**

| Agrupación de requisitos funcionales | Nombre de los requisitos funcionales |
|---|---|
| **Gestionar nomenclador 1.0** | RF 1.1 Adicionar nomenclador |
| | RF 1.2 Modificar nomenclador |
| | RF 1.3 Eliminar nomenclador |
| **Gestionar patrones 2.0** | RF 2.1 Adicionar patrón |
| | RF 2.2 Modificar patrón |
| | RF 2.3 Eliminar patrón |
| | **RF 3.0 Cargar fichero** |

| | |
|---|---|
| **Analizar fichero 4.0** | RF 4.1 Revisar fichero<br>RF 4.2 Revisar módulo |
| | **RF 5.0 Generar reporte** |

A continuación se muestran las historias de usuario de algunos requisitos de la herramienta:

**RF2 Gestionar patrón**

**RF 2.1. Adicionar patrón**

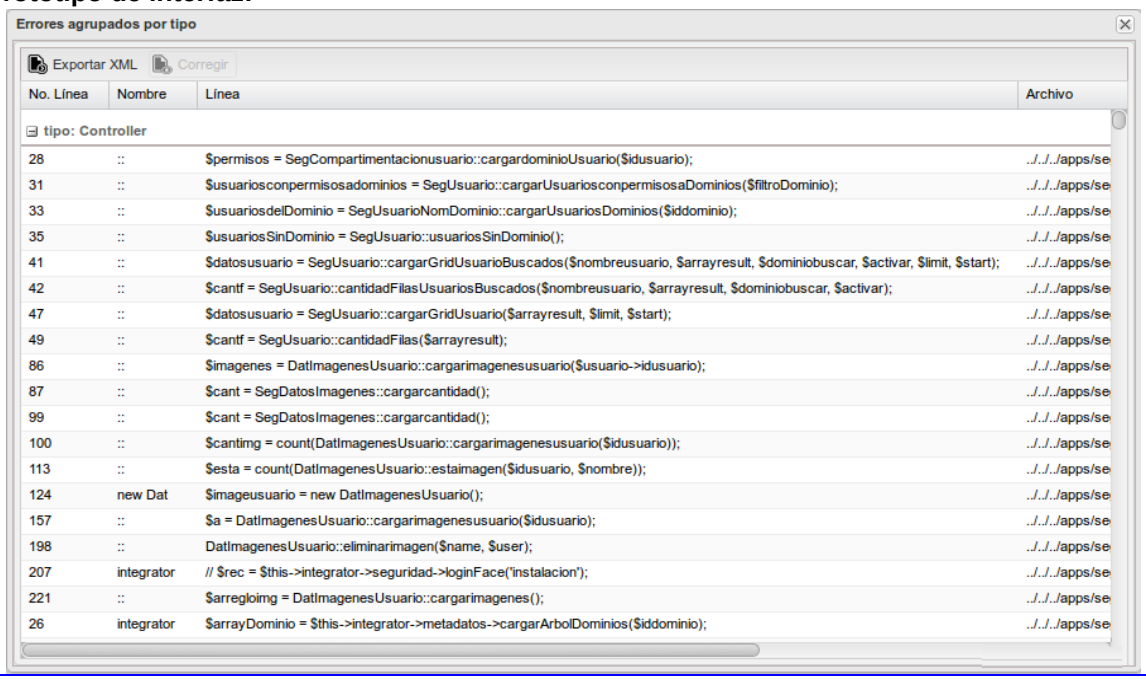
**Tabla 5. Historia de usuario correspondiente al requisito Adicionar patrón.**

| | |
|---|---|
| **Número: 2.1** | **Nombre del requisito: Adicionar patrón** |
| **Programador: Jorge Burgos Díaz** | **Iteración Asignada:** 1 |
| **Prioridad: Media** | **Tiempo Estimado:** 56 horas |
| **Riesgo en Desarrollo:**<br>Poca disponibilidad del estudiante por estar en la culminación del primer semestre de quinto año. | **Tiempo Real:** 24 horas |
| **Descripción:**<br>Añadido un nomenclador, este requisito se encarga de adicionar los patrones correspondientes al nomenclador. | |
| **Observaciones:**<br>Para adicionar un patrón debe estar previamente adicionado un nomenclador. | |
| **Prototipo de interfaz:** | |

### RF 4.2 Revisar módulo

**Tabla 6. Historia de usuario correspondiente al requisito Revisar módulo.**

| | |
|---|---|
| **Número: 4.2** | **Nombre del requisito: Revisar módulo** |
| **Programador: Jorge Burgos Díaz** | **Iteración Asignada:** 1 |
| **Prioridad:** Alta | **Tiempo Estimado:** 240 horas |
| **Riesgo en Desarrollo:** | **Tiempo Real:** 240 horas |
| **Descripción:**<br>Este requisito se encarga de revisar todos o un determinado módulo del marco de trabajo Sauxe. | |
| **Observaciones:**<br>Se debe tener un módulo cargado en la herramienta para así realizar su revisión. | |
| **Prototipo de interfaz:** | |




## 2.4.2 Requisitos no funcionales

### Restricciones de diseño

**RnF 1**: el lenguaje de programación a utilizar para desarrollar la herramienta debe ser PHP para la capa de negocio y ExtJS para la capa de presentación.

**RnF 2:** las herramientas a utilizar para desarrollar el sistema deben ser PostgreSQL 9.0 como gestor de base de datos y PgAdminIII como cliente, Doctrine para la capa de acceso a datos y Zend Framework para la capa de negocio.

**Portabilidad**

**RnF 3**: el sistema debe ser multiplataforma, haciendo énfasis en Linux y Windows.

## 2.5 Arquitectura del software

Para el desarrollo del marco de trabajo Sauxe, en el departamento de Desarrollo de Componente del CEIGE se definió el **Estilo N capas**, implementando el patrón arquitectónico Modelo-Vista-Controlador, el cual se describe a continuación: (Baryolo, 2010)

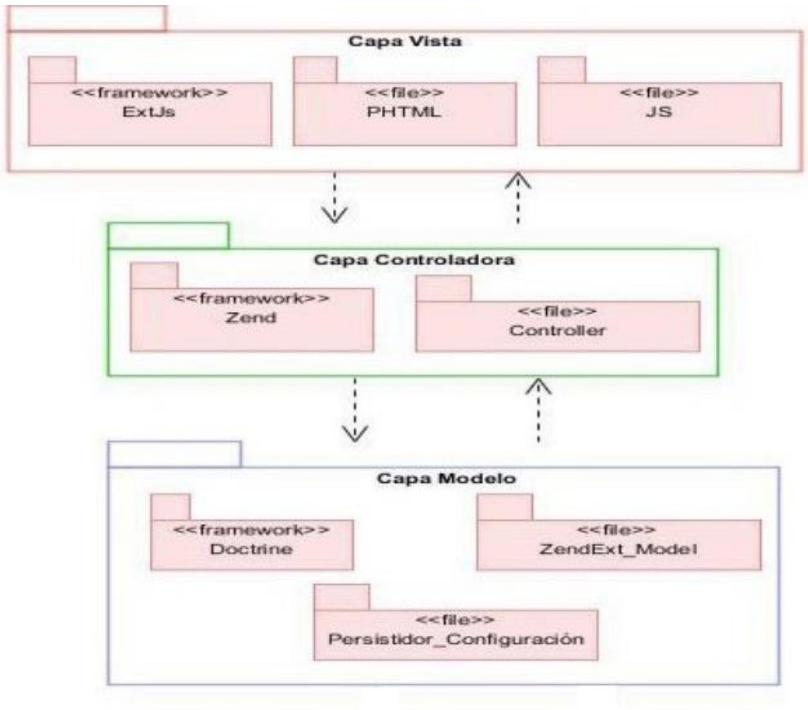



**Figura 3. Patrón arquitectónico MVC.**

El **patrón arquitectónico** MVC se encarga de separar los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos: (Baryolo, 2010)

- ✓ **Modelo:** está compuesto por datos, reglas de negocio y las funcionalidades correspondientes para la comunicación con el ORM Doctrine encargado de persistir los datos.
- ✓ **Controlador:** gestiona las peticiones del usuario y las respuestas del sistema.
- ✓ **Vista:** se encarga de elaborar y mostrar las interfaces a los usuarios. Está compuesta principalmente por el marco de trabajo ExtJS y centra su desarrollo en dos componentes fundamentales: archivos JS y archivos CSS.

A continuación se enuncian los diferentes patrones de diseño utilizados durante la fase de análisis y diseño de la herramienta.

## 2.6 Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular. (Buschmann, 1996)

Los patrones de diseño se dividen en dos vertientes, los patrones **GRASP** [5] y los **GoF**[6].

**Patrones GRASP:** describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. (Tedechi, 2015)

- ✓ **Experto:** se evidencia el uso de este patrón en todas las clases a utilizar en la herramienta pues cada clase conoce su información y es la encargada de implementar las funcionalidades que les corresponde como por ejemplo la clase *GestnomController*.
- ✓ **Creador:** su uso se evidencia en las clases controladoras pues cada una de ellas se encarga de la creación de objetos de varias clases como son *NomTipoViolacionModel*, *NomViolacionPatronesModel*, entre otras.
- ✓ **Controlador:** las diferentes clases controladoras se encargan de llevar el control de todos los eventos relacionados con el negocio. Implementan las funcionalidades que dan respuesta a las peticiones del usuario como por ejemplo la clase controladora *GestnomController*.
- ✓ **Alta Cohesión:** indica que la información almacenada en las clases debe ser coherente y relacionada a lo que se maneja en dicha clase, se evidencia su uso en todas las clases como por ejemplo en *GestnomController*.
- ✓ **Bajo Acoplamiento:** se evidencia en la poca relación existente entre las clases que conforman la herramienta.

**Patrones GoF:** se dieron a conocer a principios de los años 90 con el libro *"Patrones de Diseño. Elementos de Software Reusable Orientado a Objetos"*. En dicho libro se hace una recopilación de 23 patrones de diseño comunes, clasificados en tres grupos de acuerdo a su naturaleza:

- ✓ **Patrones Creacionales:** inicialización y configuración de objetos. (Tedechi, 2015)
- ✓ **Patrones Estructurales:** separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes. (Tedechi, 2015)
- ✓ **Patrones de Comportamiento:** más que describir objetos o clases, describen la comunicación entre ellos. (Tedechi, 2015)

[5] GRASP: por las siglas en inglés de General Responsibility Assignment Software Patterns.

[6] GoF: por sus siglas en inglés de Gang of Four,en español Banda de los Cuatro

Para el desarrollo de la herramienta se utilizó el patrón creacional **Singleton o Solitario** el cual garantiza que una clase tiene una única instancia, proporcionando un punto de acceso global a la misma. (Larman, 2003) El uso del patrón se evidencia en la clase *NomViolacionPatrones* la cual se encarga de la comunicación con la capa de acceso a datos.

## 2.7 Modelo de diseño

Los diagramas de clases especifican las diferentes clases que serán utilizadas en el sistema, así como las relaciones que existen entre ellas. Para la herramienta se elaboró el diagrama de clases del diseño con estereotipos web el cual especifica las diferentes clases que serán utilizadas y las relaciones que existen entre ellas. A continuación se muestra el diagrama:

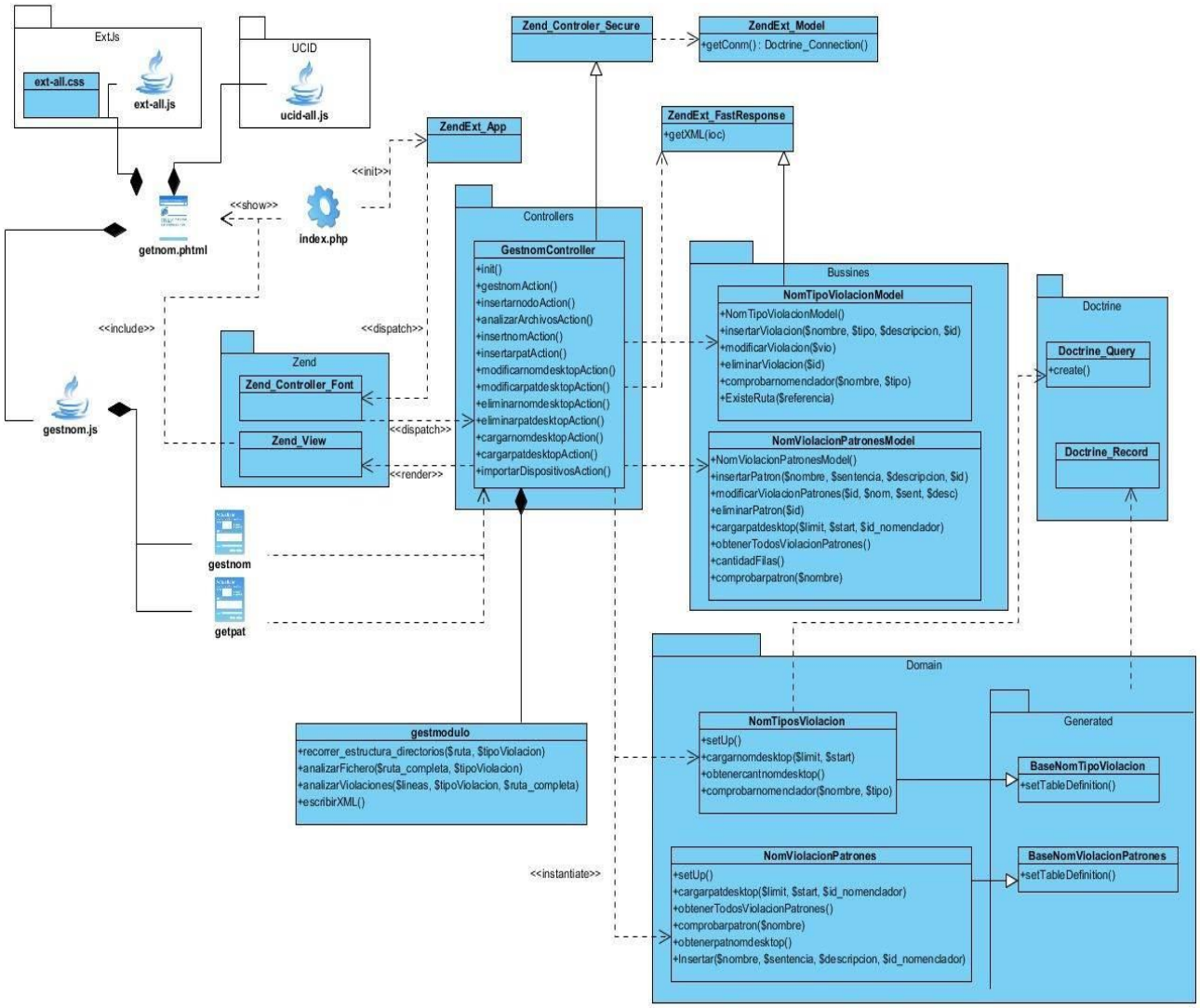



**Figura 4. Diagrama de clases del diseño con estereotipos web.**

## 2.8 Modelo de datos

Otra de las etapas del diseño es la elaboración del **modelo de datos**, según C.J Date se definen como una definición lógica, independiente y abstracta de los objetos, operadores y demás que en conjunto constituyen la máquina abstracta con la que interactúan los usuarios. Los objetos permiten modelar la estructura de los datos y los operadores permiten modelar su comportamiento. (Date, 2001). El modelo de datos correspondiente a la herramienta contiene dos tablas que se muestran a continuación:

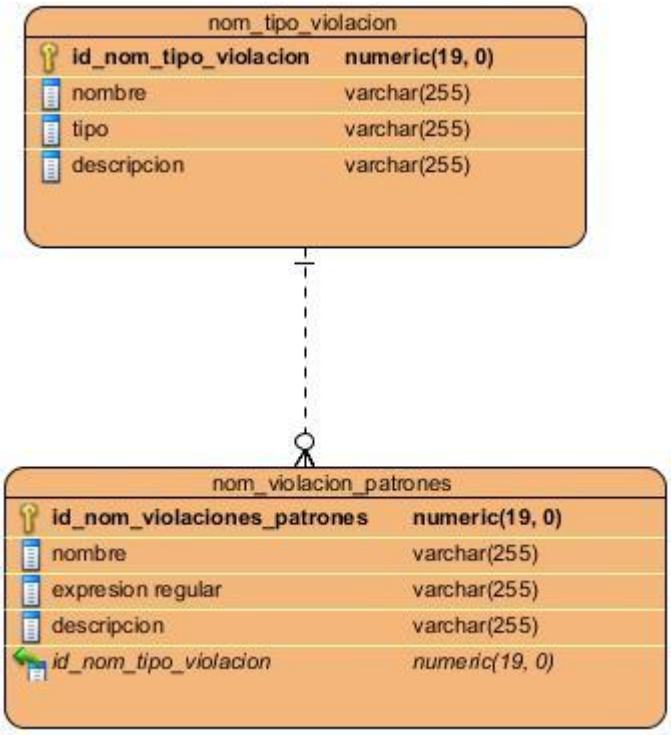



**Figura 5. Modelo de datos.**

A continuación se describen los atributos de las siguientes tablas:

**Tabla: *nom_tipo_violacion***

- ***nombre*:** nombre de la capa que se va a agregar.
- ***tipo*:** tipo de clase que sea (controladora, modelo o domain).
- ***descripcion:*** observación del nomenclador.

**Tabla: *nom_violacion_patrones***

- ***id_nom_violacion_patrones*:** llave primaria de la tabla.
- ***nombre*:** nombre de la violación.
- ***expresion regular*:** expresión regular de la violación.
- ***descripcion*:** observación del patrón.
- ***id_nom_tipo_violacion*:** llave foránea que determina a qué violación pertenece.

## 2.9 Resultados obtenidos de la aplicación de las métricas para validar el diseño

### Métricas de software

Las métricas de software constituyen una medida cuantitativa que permite a los desarrolladores tener una visión profunda de la eficacia del proceso de software. En esta evaluación se reúnen los datos básicos referentes a los factores de calidad que posteriormente serán analizados, comparados con promedios anteriores y evaluados, para determinar las mejorías en términos de estos factores. (Pressman, 2005)

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

- ✓ **Responsabilidad:** consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta. (Baryolo, 2010)
- ✓ **Complejidad de implementación:** consiste en el grado de complejidad que posee la implementación de un diseño de clases específico. (Baryolo, 2010)
- ✓ **Reutilización:** consiste en el grado de reutilización presente en una clase o estructura de clases, dentro de un diseño de software. (Baryolo, 2010)
- ✓ **Acoplamiento:** las conexiones físicas entre los elementos del diseño orientado a objeto, representan el acoplamiento dentro de un sistema orientado a objeto. (Pressman, 2005)
- ✓ **Complejidad del mantenimiento:** 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. (Baryolo, 2010)
- ✓ **Cantidad de pruebas:** número o grado de esfuerzo para realizar las pruebas de calidad del producto diseñado. (Baryolo, 2010)

La métrica **Tamaño Operacional de Clases (TOC)** se encarga de contar el número de métodos u operaciones (de instancia privada y heredada) que están encapsulados dentro o por una clase. Evalúa los siguientes atributos de calidad: (Baryolo, 2010)

**Tabla 7. Atributos de calidad evaluados por la métrica TOC.**

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

Para la evaluación de dichos atributos de calidad, se definen los siguientes criterios y categorías de evaluación:

**Tabla 8. Criterios de evaluación para la métrica TOC.**

| Atributo | Categoría | Criterio |
|---|---|---|
| **Responsabilidad** | Baja | <=promedio |
| | Media | Promedio< y< 2*promedio |
| | Alta | >2*promedio |
| **Complejidad de implementación** | Baja | <=promedio |
| | Media | Promedio< y< 2*promedio |
| | Alta | >2*promedio |
| **Reutilización** | Baja | >2*promedio |
| | Media | Promedio< y< 2*promedio |
| | Alta | <=promedio |

La métrica **Relaciones entre Clases (RC)** se encarga de contar el número de relaciones de uso de una clase con otra. (Baryolo, 2010) La relación entre esta métrica y los atributos de calidad, se observa en la Tabla 9.

**Tabla 9. Relaciones entre clases (RC).**

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

Para la evaluación de dichos atributos de calidad, se definieron los criterios y categorías de evaluación:

**Tabla 10. Criterios de evaluación para la métrica RC.**

| Atributo | Categoría | Criterio |
|---|---|---|
| **Acoplamiento** | Ninguno | 0 |
| | Baja | 1 |
| | Media | 2 |

| | | |
|---|---|---|
| | Alta | >2 |
| **Complejidad de mantenimiento** | Baja | <=promedio |
| | Media | Promedio< y< 2*promedio |
| | Alta | >2*promedio |
| **Reutilización** | Baja | >2*promedio |
| | Media | Promedio< y< 2*promedio |
| | Alta | <=promedio |
| **Cantidad de pruebas** | Baja | <=promedio |
| | Media | Promedio< y< 2*promedio |
| | Alta | >2*promedio |

**Tamaño Operacional de Clases (TOC)**

Ver los instrumentos y la tabla de resultados para la métrica TOC en el Anexo 1.

La Figura 6 muestra la representación de los resultados obtenidos agrupados en los intervalos definidos. El gráfico refleja que la mitad de las clases tienen de 1 a 5 procedimientos. Esto demuestra que el funcionamiento general de la herramienta está distribuido equitativamente entre las diferentes clases.

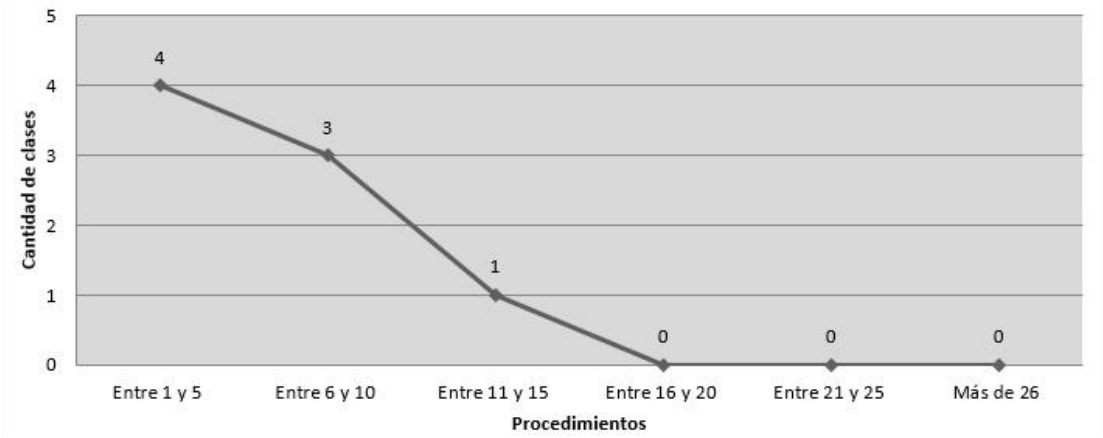



**Figura 6. Representación de la evaluación de la métrica TOC.**

La Figura 7 muestra los resultados obtenidos en el instrumento en porciento agrupados en los intervalos definidos.

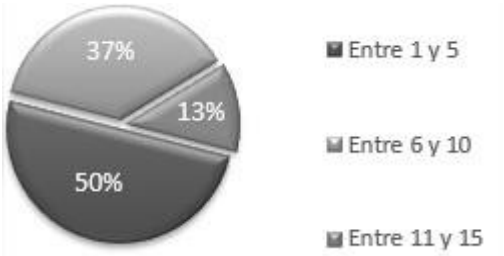



**Figura 7. Representación en porciento de los resultados obtenidos en la evaluación de la métrica TOC.**

En la Figura 8 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad. Este gráfico muestra un resultado satisfactorio pues el 50% de las clases poseen una baja responsabilidad. Esta característica permite que en caso de fallos, como la responsabilidad está distribuida de forma equilibrada ninguna clase es demasiado crítica como para dejar al sistema fuera de servicio.


**Figura 8. Resultados de la evaluación de la métrica TOC para el atributo responsabilidad.**

En la Figura 9 se observa la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo complejidad de implementación. Este gráfico muestra un resultado satisfactorio pues el 50% de las clases poseen una baja complejidad de implementación. Esta característica permite mejorar el mantenimiento y soporte de estas clases.


**Figura 9. Resultados de la evaluación de la métrica TOC para el atributo complejidad de implementación.**

La Figura 10 muestra la representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo reutilización. El diseño de la herramienta tiene un grado de eficiencia aceptable pues solamente el 10% del total de las clases poseen una baja reutilización.


**Figura 10. Resultados de la evaluación de la métrica TOC para el atributo reutilización.**

Analizando los resultados obtenidos de la métrica TOC, se puede concluir que el diseño de la herramienta tiene una calidad aceptable teniendo en cuenta los resultados arrojados por los atributos analizados. Solo el 13% de las clases presentan una alta responsabilidad y alta complejidad de implementación y un 10 % poseen una baja reutilización, lo cual demuestra que el resultado es satisfactorio.

**Relaciones entre Clases (RC)**

El gráfico de la Figura 11 refleja que el 15% de las clases tienen más de tres dependencias, y el 62% una dependencia con otra clase, demostrando que la mayoría de las clases presentan niveles aceptables de calidad.

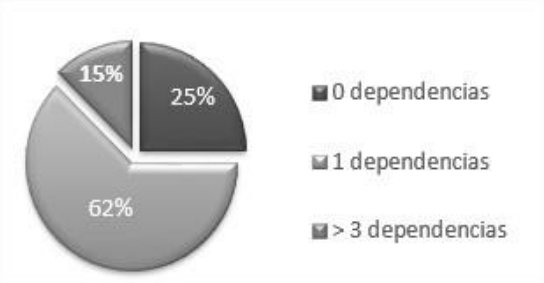



**Figura 11. Representación en porciento de los resultados obtenidos en los intervalos definidos según la métrica RC.**

La Figura 12 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo acoplamiento. Se evidencia un bajo acoplamiento entre las clases pues el 25% de las clases no presentan dependencias con otra y el 62% una dependencia con otra. Este resultado es muy favorable para el diseño de la herramienta pues al existir poca dependencia entre las clases aumenta el grado de reutilización de la herramienta.


**Figura 12. Resultados de la evaluación de la métrica RC para el atributo acoplamiento.**

La Figura 13 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo complejidad de mantenimiento. El gráfico refleja un resultado aceptable del atributo pues el 87% de las clases presentan una baja complejidad de mantenimiento.


**Figura 13. Resultados de la evaluación de la métrica RC para el atributo complejidad de mantenimiento.**

La Figura 14 muestra la representación de la incidencia de los resultados de la evaluación del atributo reutilización. Esto evidencia que el 88% de las clases poseen una alta reutilización lo que es un factor fundamental que debe ser tenido en cuenta en el desarrollo de software.


**Figura 14. Resultados de la evaluación de la métrica RC para el atributo reutilización.**

La Figura 15 muestra la representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo cantidad de pruebas.

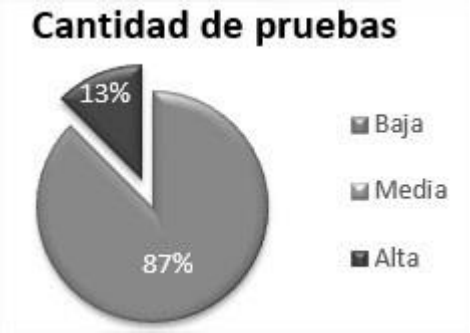



**Figura 15. Resultados de la evaluación de la métrica RC para el atributo cantidad de pruebas.**

Analizando los resultados de la evaluación del instrumento de medición de la métrica relación entre clases (RC), se puede concluir que el diseño posee una calidad aceptable, ya que el 85 % de las clases de la solución tienen menos de dos dependencias con otras clases. También se refleja en los resultados que en el 87 % el acoplamiento entre clases es mínimo; que la complejidad de mantenimiento, la cantidad de pruebas son bajas en un 87 % y la reutilización de

las clases son altas en un 88 %. Este resultado demuestra que los atributos de calidad se encuentran en niveles satisfactorios. Los instrumentos y las tablas de resultados del instrumento de medición de la métrica RC se pueden observar en el Anexo 2 del documento.

**Conclusiones del capítulo**

En este capítulo se enunciaron los aspectos fundamentales que se llevan a cabo durante la disciplina de análisis y diseño para dar paso a la implementación de la herramienta. En el modelo conceptual mostrado se puntualizan los conceptos relacionados con el dominio del problema y su representación conforma una guía visual para la implementación de la solución tributando directamente a la disciplina de requisitos.

La captura y validación de los 10 requisitos funcionales aportó una explicación escrita del problema y una mayor comprensión del comportamiento que se necesita implementar en la herramienta. Los requisitos no funcionales que debe cumplir la herramienta, garantizan que su desempeño es el adecuado para su integración en Sauxe.

La utilización de estilos y patrones arquitectónicos brinda solidez a la arquitectura, garantizando que la estructura propuesta rija de forma correcta el posterior diseño, a la vez que los patrones de diseño especificados permiten ganar en reutilización y brindan robustez a la solución.

La realización del diagrama de clases del diseño con estereotipos web y el modelo de datos, permitió representar las clases y relaciones que componen la herramienta; el flujo del comportamiento e interacción entre los objetos de la aplicación y la estructura que tendrán los datos de la herramienta respectivamente.

Se realizó la validación del diseño empleando para ello las métricas de diseño, donde se evidenciaron niveles satisfactorios como el 62 % de baja responsabilidad y un grado de reutilización alto en un 88 % de las clases de la herramienta, así como los niveles de los restantes indicadores evaluados.

# Capítulo 3: Implementación y pruebas

## 3.1 Introducción

El siguiente capítulo aborda los elementos relacionados con la implementación de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe, así como lo referente a las pruebas realizadas a la herramienta para validar su correcto funcionamiento. Se describen los estándares de codificación utilizados en función de garantizar el entendimiento del código fuente. Se muestran los casos de pruebas diseñados para realizar las pruebas funcionales, así como los resultados obtenidos, validando así que los requisitos identificados fueron implementados correctamente.

## 3.2 Modelo de implementación

El modelo de implementación se inicia a partir de los resultados obtenidos en el diseño y describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado. También describe cómo dependen los componentes unos de otros además de los recursos necesarios para poder ejecutar la herramienta desarrollada. (Acuña, 2013)

### 3.2.1 Diagrama de componentes

Uno de los productos de trabajo generados en esta fase es el diagrama de componentes, el cual representa la estructura física de la herramienta, su agrupación por paquetes y las dependencias entre estos. A continuación se muestra el diagrama de componentes elaborado.

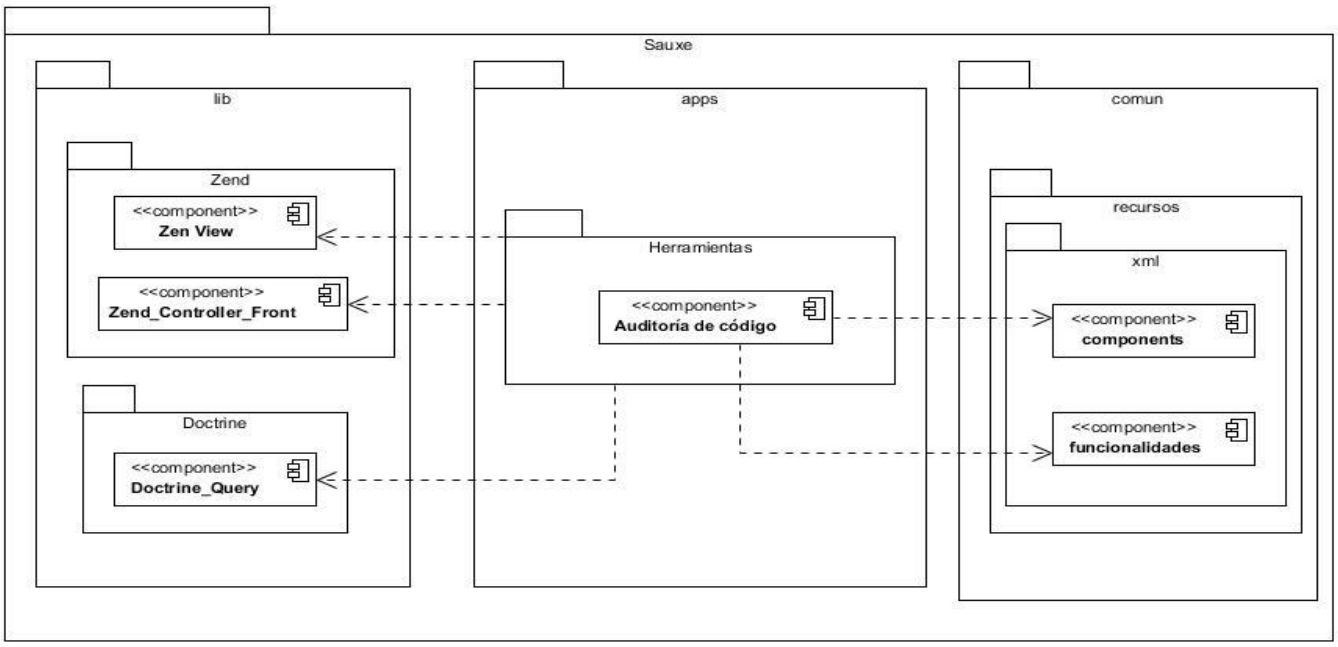



**Figura 16. Diagrama de componentes.**

### 3.3 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. El uso de estándares de codificación permite lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su mantenibilidad a lo largo del tiempo. (Cuba, 2008)

Para el desarrollo de la herramienta se utilizarán algunos de los estándares de codificación y normas propuestos como parte de la línea de arquitectura determinada para el desarrollo del ERP[7] Cuba. (Cuba, 2008)

Los estándares utilizados en la codificación fueron los siguientes:

- ✓ **Notación Húngara:** 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 indique su tipo de dato o ámbito. (Sperberg, 2012)
- ✓ **Notación PascalCasing:** es como la notación húngara pero sin prefijos. En este caso los identificadores y nombres de las variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula. (Svensk, 2012)
- ✓ **Notación CamelCasing:** es parecido al PascalCasing con la excepción que la letra inicial del identificador no debe estar en mayúscula. (Svensk, 2012)

**Nomenclatura de las clases**

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing.

Ejemplo: *NomTipoViolacion*.

**Nomenclatura según el tipo de clases**

- ✓ **Clase controladora:** después del nombre llevan la palabra "*Controller*".
  Ejemplo: *GestnomController*.

**Clases del modelo:**

- ✓ **business (Negocio):** las clases que se encuentran dentro de business después del nombre llevan la palabra "*Model*".
  Ejemplo: *NomTipoViolacionModel.*
- ✓ **domain (Dominio):** el nombre que reciben las clases que se encuentran dentro de *domain* es el de la tabla en la base de datos.

[7] **ERP:** por las siglas en inglés **E**nterprise **R**esource **P**lanning.

Ejemplo: *NomTipoViolacion.*

- ✓ **generated:** el nombre que reciben las clases que se encuentran dentro de generated comienza con el prefijo "*Base*" y después el nombre de la tabla en la base de datos. Ejemplo: *BaseNomTipoViolacion*.

**Nomenclatura de las funciones**

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, y en caso de ser una acción de la clase controladora se debe especificar el nombre de dicha acción en minúscula y seguido el sufijo "*Action*".

Ejemplo: *insertarnomAction*.

**Nomenclatura de las variables**

El nombre a emplear para las variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing.

Ejemplo: *$tipo*.

**Nomenclatura de los comentarios**

Se debe comentar todo lo que se haga dentro del desarrollo, establecer las pautas que conlleven a lograr un código más legible y reutilizable y así se puede aumentar su mantenimiento a lo largo del tiempo. Los comentarios deben ser lo bastante claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando.

**Estilo de código**

En la implementación, al escribir las sentencias en php, se utilizarán los tabs del lenguaje como se muestra en la siguiente figura.


**Figura 17. Estilo de código.**

- ✓ **Sangría o indexado:** la política de sangría a utilizar en la implementación es por **tab**. Ejemplo:

```php
<?php
/**
 * Indentation
 */
class Example {
    var $theInt = 1;
    function foo($a, $b) {
    switch ( $a) {
    case 0 :
    $Other->doFoo ();
    break;
    default :
    $Other->doBaz ();
    }
    }
    function bar($v) {
    for($i = 0; $i < 10; $i ++) {
    $v->add ( $i );
    }
    }
}
?>
```

**Figura 18. Estilo de código: Sangría e indexado.**

- ✓ **Brazas o llaves:** en la declaración de clases o interfaces, métodos, bloques y switch, la apertura de llaves se hace en la misma línea.

```php
<?php
/**
 * Braces
 */
interface EmptyInterface {
}

class Example {
    function bar($p) {
        for($i = 0; $i < 10; $i ++) {
        }
        switch ( $p) {
            case 0 :
                $fField->set ( 0 );
            break;
            case 1 :
                {
                    break;
                }
            default :
                $fField->reset ();
        }
    }
}
?>
```

**Figura 19. Estilos de código: Brazas o llaves.**

### 3.4 Pruebas de software

Pressman, declara que las pruebas de software son un elemento crucial para garantizar la calidad del producto y permiten validar las especificaciones, el diseño y la programación. Estas tienen como objetivo, además de descubrir errores, medir el grado en que el software cumple con los requerimientos definidos. (Pressman, 2005) Existen dos vertientes de pruebas, las pruebas de caja negra y las pruebas de caja blanca.

Las **pruebas de caja negra**, permiten obtener conjuntos de entradas que ejerciten los requisitos funcionales del software, complementándose con las pruebas de caja blanca, con el objetivo de demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto. (Pressman, 2005)

La prueba de caja negra para Pressman intenta encontrar errores de las siguientes categorías:

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

Para las pruebas de caja negra existen varias técnicas, algunas de ellas son:

- ✓ **Partición equivalente:** permite examinar los valores válidos e inválidos de las entradas existentes en el software. Además descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. Esto permite reducir el número de casos de prueba a elaborar. (Pressman, 2005)
- ✓ **Análisis de valores límites:** los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta es una técnica que complementa la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, lleva a elección de casos de prueba en los extremos de la clase. (Pressman, 2005)
- ✓ **Prueba de comparación:** este tipo de pruebas se emplea cuando la fiabilidad del software es algo crítica (por ejemplo cuando se desarrolla para aeronaves o plantas nucleares) varios equipos de ingeniería del software desarrollan versiones independientes de la misma aplicación, usando los mismos requisitos. Todas las versiones son probadas con los mismos datos, para asegurar que todas proporcionan una salida idéntica. (Pressman, 2005)
- ✓ **Prueba de la tabla ortogonal:** esta prueba puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para solicitar pruebas exhaustivas. El método de la tabla ortogonal es útil al encontrar errores asociados con fallos localizados. (Pressman, 2005)

Analizadas todas estas técnicas se concluye que todas son útiles siempre que se escoja bien el contexto en el cual se van a aplicar. Cada proyecto de software tiene características muy particulares. Otro tipo de pruebas que se aplica con frecuencia al software son las de caja blanca, que de acuerdo con Pressman en (Pressman, 2005), las **pruebas de caja blanca**, es un método que usa la estructura de control del diseño procedimental.

Mediante este método se pueden obtener casos de prueba que:

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

Para la realización de estas pruebas existen diferentes métodos que se analizarán a continuación:

- ✓ **Camino básico:** permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Para obtener el conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los casos de pruebas obtenidos garantizan que se ejecute al menos una vez cada sentencia del programa. (Pressman, 2005)
- ✓ **Prueba de condición:** es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa. (Pressman, 2005)
- ✓ **Pruebas de flujos de datos:** selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variantes del programa. (Pressman, 2005)
- ✓ **Prueba de bucles:** se centra exclusivamente en la validez de las construcciones de bucles. Se pueden definir cuatro clases diferentes de bucles: bucles simples, bucles concatenados, bucles anidados y bucles no estructurados. (Pressman, 2005)

**Pruebas de caja blanca:** la técnica utilizada para la aplicación de esta prueba fue la del camino básico. El fragmento del código utilizado se muestra a continuación:

```
function modificarPatron() {
    if (regPatrones.getForm().isValid()) {                                  1
        var dMod = denMod != Ext.getCmp('nombreP').getValue();              2
        var aMod = abrevMod != Ext.getCmp('sentencia').getValue();
        var deMod = desMod != Ext.getCmp('descripcionP').getValue();
        if (dMod || aMod || deMod) {                                        3
            regPatrones.getForm().submit({                                  4
                url: 'modificarpatdesktop',
                waitMsg: perfil.etiquetas.lbMsgFunModificarPatMsg,
                params: {
                    id: smPatrones.getLastSelected().data.id_nom_violacion_patrones
                },
                failure: function(form, action) {
                    if (action.result.codMsg != 3) {                        5
                        winModpat.hide();
                        mostrarMensaje(1, 'Se ha modificado el patrón satisfactoriamente.');    6
                        stPatrones.reload();
                    }                                                       7

                }
            });
        }
        else
            mostrarMensaje(3, perfil.etiquetas.NoModifyP);                  8
    }
    else
        mostrarMensaje(3, perfil.etiquetas.lbMsgErrorEnCamops);             9
}                                                                           10
```

**Figura 20. Código fuente utilizado para realizar la prueba de Caja blanca.**

A continuación se enumeran las sentencias de código del método ***modificarPatron()*** a modo de ejemplo.

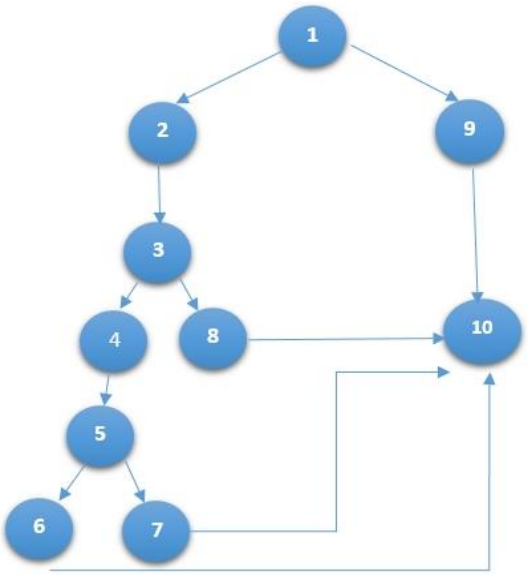



**Figura 21. Grafo de flujo asociado a la funcionalidad modificarPatron().**

La complejidad ciclomática es la métrica de software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Esta métrica calcula la cantidad de caminos independientes de cada una de las funcionalidades del programa. También provee el límite superior para el

número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. (Pressman, 2005)

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad sea el correcto.

1. **V(G) = R** donde R representa la cantidad total de regiones.

   V(G) = 4

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

   V(G) = 12 – 10 + 2

   V(G) = 4

3. **V (G) = P + 1** donde **P** es el número de nodos predicados contenidos en el grafo de flujo (se denomina nodo predicado a los nodos de los cuales parten dos o más aristas).

   V(G) = 3 + 1

   V(G) = 4

Dado que el cálculo de las tres fórmulas anteriormente mencionadas arrojó el mismo resultado se puede plantear que la complejidad ciclomática del método es cuatro. Esto significa que existen cuatro posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

- ✓ **Camino básico #1:** 1 – 9 – 10.
- ✓ **Camino básico #2:** 1 – 2 – 3 – 8 – 10.
- ✓ **Camino básico #3:** 1 – 2 – 3 – 4 – 5 – 7 – 10.
- ✓ **Camino básico #4:** 1 – 2 – 3 – 4 – 5 – 6 – 10.

Para cada camino básico determinado se realiza un diseño de caso de prueba:

- ✓ **Caso de prueba para el camino básico #1:** Si regPatrones.getForm().isValid() == false.
- ✓ **Caso de prueba para el camino básico #2:** Si regPatrones.getForm().isValid() == true, Si dMod || aMod || deMod == false.
- ✓ **Caso de prueba para el camino básico #3:** Si regPatrones.getForm().isValid() == true, Si dMod || aMod || deMod == true, Si action.result.codMsg =3.
- ✓ **Caso de prueba para el camino básico #4:** Si regPatrones.getForm().isValid() == true, Si dMod || aMod || deMod == true, Si action.result.codMsg !=3.

**Pruebas de caja negra**

Para evaluar los requisitos funcionales de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe se realizaron pruebas de caja negra en la fase de pruebas internas. Para aplicarlas se confeccionaron los diseños de caso de prueba para cada una de las funcionalidades de la herramienta, los cuales pueden ser consultados en los productos de trabajo.

Para comprobar la calidad de la herramienta se realizaron dos iteraciones para las revisiones por un grupo de profesionales del departamento de Desarrollo de Componente. Las no conformidades encontradas se clasificaron en:

- ✓ No conformidades detectadas en la documentación.
- ✓ No conformidades detectadas en la aplicación.

**Resultados de las pruebas**

En la Tabla 11 se recoge la cantidad de no conformidades detectadas en la documentación y la aplicación por cada iteración.

**Tabla 11. No conformidades detectadas en la documentación y la aplicación por iteraciones.**

| Tipo de no conformidades | Documentación | Aplicación |
|---|---|---|
| **Primera iteración** | | |
| **Significativa** | 0 | 4 |
| **No significativa** | 5 | 3 |
| **Total** | 5 | 7 |
| **Segunda iteración** | | |
| **Significativa** | 0 | 0 |
| **No significativa** | 0 | 0 |
| **Total** | 0 | 0 |

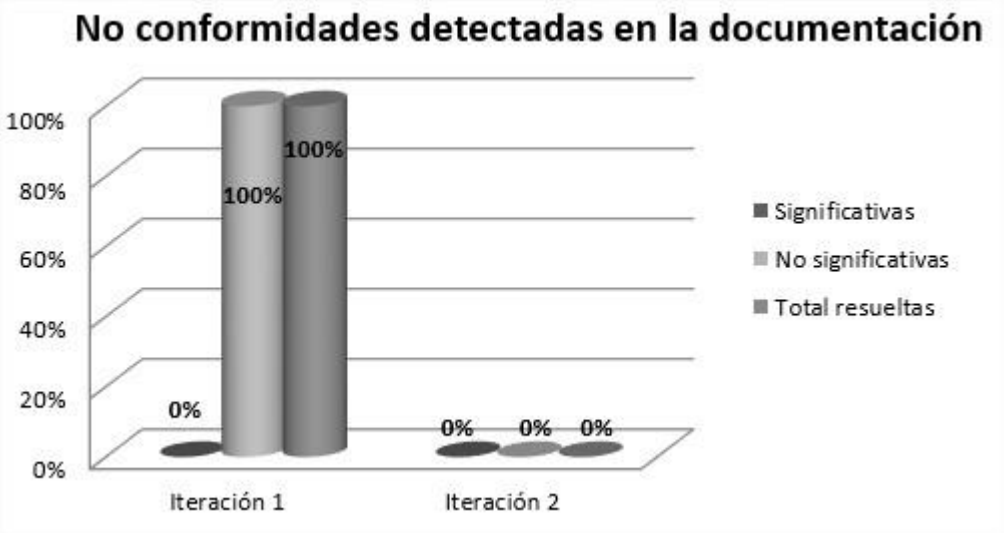



**Figura 22. Porciento que representan las no conformidades detectadas en la documentación por cada una de las iteraciones.**

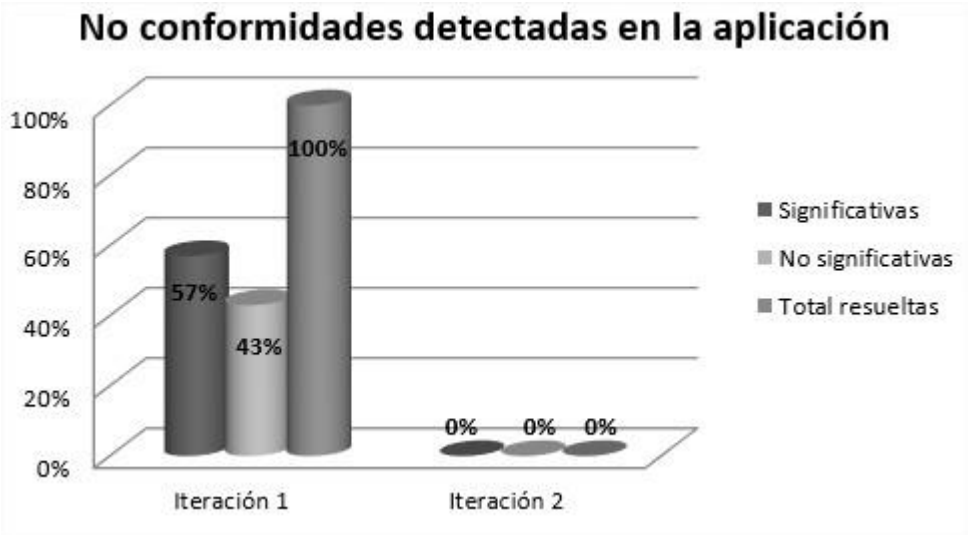



**Figura 23. Porciento que representan las no conformidades detectadas en la aplicación por cada una de las iteraciones.**

### 3.5 Pruebas de aceptación

Según la Ing. Tamara Sánchez en (Sánchez, 2014), la disciplina **Pruebas de Aceptación** es una prueba final que tiene como objetivo verificar que el software realizado está listo y que puede ser usado por los usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido.

Para verificar la herramienta se realizaron dos iteraciones por el arquitecto de datos en el departamento de Desarrollo de Componente, el Ing. Inoelkis Velazquez Osorio. En la verificación se encontraron seis no conformidades mencionadas a continuación:

- ✓ errores en las funcionalidades de la herramienta.
- ✓ errores en la interfaz de usuario.

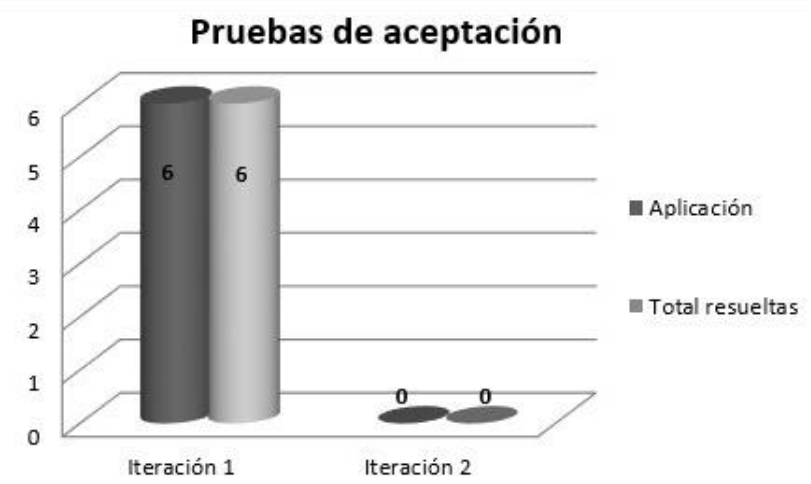



**Figura 24. Representación de los resultados de las pruebas de aceptación.**

Posteriormente las no conformidades detectadas en la aplicación fueron resueltas quedando la herramienta en la segunda iteración lista para ser usuada por los usuarios finales.

### 3.6 Validación de la idea a defender

En el marco de trabajo Sauxe no existía una correcta utilización de los estándares arquitectónicos definidos por el patrón MVC. La falta de conocimiento sobre estos estándares trae como consecuencia que se violen los esquemas de organización de código, realizando productos incompatibles con este patrón. Esto trae consigo que se debía revisar el código fuente sistemáticamente de forma manual en busca de estas violaciones y a su vez reorganizando el código, por lo que este proceso era muy engorroso.

Con el objetivo de aportar una solución factible a este problema se decide que si se desarrolla una herramienta para el tratamiento de violaciones de los estándares arquitectónicos definidos en el marco de trabajo Sauxe, ocurrirá un aumento en la detección de violaciones arquitectónicas en las aplicaciones desarrolladas con el marco de trabajo Sauxe así como la disminución del tiempo de reorganización del código fuente teniendo en cuenta los estándares definidos. Con el objetivo de validar el cumplimiento de la idea a defender para la herramienta, se confeccionó un cuestionario para ser respondido por tres integrantes del departamento de Desarrollo de Componente, los mismos son:

- ✓ Analista principal, Ing. Katia Saria Preval.
- ✓ Arquitecto de datos, Ing. Inoelkis Velazquez Osorio.
- ✓ Jefe del departamento, Ing. René R. Bauta Camejo.

En el cuestionario se analizaron en total nueve clases funcionales del marco de trabajo Sauxe v2.3.0. Las clases seleccionadas atendiendo a su complejidad son:

- ✓ **Complejidad Alta:** las clases *GestSistemaController*, *DatSistemaModel* y *DatSistema*. Estas responden a la funcionalidad Gestionar sistema del sistema de seguridad Acaxia.

Para configurar la seguridad en un nuevo componente es necesario registrarlo mediante esta funcionalidad, así como configurar la conexión a base de datos. La complejidad de implementación es alta por el alto grado de responsabilidad que presenta esta funcionalidad. Estas clases presentan un total de 1 886 sentencias y 84 funcionalidades.

- ✓ **Complejidad Media:** las clases *GestFuncionalidadController*, *DatFuncionalidadModel* y *DatFuncionalidad*. Estas responden a la funcionalidad Gestionar funcionalidad del sistema Acaxia. Luego de registrado el sistema se procede a registrar las funcionalidades con que cuenta. Su complejidad en la implementación no es crítica como la funcionalidad anterior pero presenta un número considerable de funciones y sentencias. Estas clases presentan un total de 455 sentencias con 38 funcionalidades.
- ✓ **Complejidad Baja**: las clases *GestNomIdiomaController***,** *NomIdiomaModel* y *NomIdioma*. Estas responden a la funcionalidad Gestionar idioma del sistema Acaxia. Al definir un usuario uno de los campos a especificar es el idioma a utilizar cuando inicie en el sistema. Esta funcionalidad presenta complejidad baja debido a que no representa dificultad en su implementación. La misma cuenta con 207 sentencias y 20 funcionalidades.

En las figuras siguientes se muestran los resultados obtenidos:


**Figura 25. Representación gráfica de los resultados de la cantidad de violaciones detectadas.**


**Figura 26. Representación gráfica de los resultados del tiempo empleado en la detección de las violaciones de los estándares arquitectónicos.**


**Figura 27. Representación gráfica de los resultados en el tiempo empleado en la reorganización de código.**

En las gráficas anteriores se puede observar que la cantidad de violaciones detectadas en las clases seleccionadas varían, ya que con la herramienta se detectaron 33 violaciones que no fueron detectadas manualmente. Los resultados del cuestionario revelan que de forma manual, la detección de violaciones tiene una duración aproximada a 1,33 horas, para una sola persona. Haciendo uso de la herramienta, este proceso tiene una duración de 0,001667 horas, para una reducción del 99.87 % del tiempo de detección de violaciones. Por otra parte, el tiempo que se empleó para la reorganización de código de forma manual fue completado en 29,5 horas (tres jornadas laborales de ocho horas). Sin embargo apoyándose en la herramienta esta tarea se puede realizar en 7,5 horas (una jornada laboral). Con el cuestionario aplicado se ve reflejado en los resultados la disminución del tiempo de reorganización en un 74,58 % y el aumento en un 22,76 % de detección de violaciones en el marco de trabajo Sauxe, dándole cumplimiento a la idea a defender planteada en la presente investigación.

**Conclusiones del capítulo**

Luego de elaborado el presente capítulo se arriba a las conclusiones:

El diagrama de componentes permitió representar mediante una colección de arcos y nodos los aspectos físicos de la herramienta a desarrollar. Las normas y estándares de codificación utilizados permitieron brindarle legibilidad y mayor entendimiento al código fuente.

Se validó la herramienta mediante pruebas funcionales entre las cuales se encuentran:

- ✓ Las pruebas estructurales realizadas al código a partir de la técnica del camino básico, evidencian una correcta implementación, que garantizó que todas las sentencias de la herramienta se ejecutaron al menos una vez.
- ✓ Se realizaron pruebas funcionales, donde con dos iteraciones totales se lograron resolver las no conformidades detectadas.
- ✓ Se realizaron las pruebas de aceptación, quedando la herramienta lista para ser utilizada por los usuarios finales.

Se validó la investigación mediante el pre-experimento donde se evidencia la disminución en un 74,58 % del tiempo de reorganización del código fuente y un aumento de 24,78 % en la detección de violaciones en las aplicaciones desarrolladas en el marco de trabajo Sauxe, dándole cumplimiento a la idea a defender planteada.

## Conclusiones generales

Luego del desarrollo de la herramienta para el tratamiento de violaciones de los estándares arquitectónicos en el marco de trabajo Sauxe, se arriban a las siguientes conclusiones:

- ✓ Se construyó el marco teórico conceptual de la investigación sobre la auditoría de código en herramientas y plugins, donde se obtuvo como resultado los conceptos fundamentales y las herramientas existentes para la revisión de código donde las mismas aportaron buenas prácticas en el desarrollo de la solución.
- ✓ Se generaron los productos de trabajo necesarios durante las disciplinas requisitos y análisis y diseño que propone la Metodología de desarrollo para la Actividad productiva de la UCI para dar paso a la disciplina de implementación.
- ✓ Se desarrolló la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe empleando herramientas y tecnologías libres obteniendo un producto funcional acorde a los requisitos identificados.
- ✓ Se realizaron pruebas para validar el diseño mediante las métricas Tamaño Operacional de Clase y Relaciones entre Clases arrojando resultados satisfactorios lo cual demuestra que el diseño realizado cumple con los parámetros de calidad establecidos por dichas métricas.
- ✓ Como constancia, la herramienta fue probada por un grupo de trabajo del departamento de Desarrollo de Componente, los cuales constataron que la herramienta cumple con los objetivos determinados al inicio de la investigación.
- ✓ Se validó la investigación mediante el pre-experimento donde se evidenció una disminución en un 74,58 % del tiempo de reorganización de código y se obtuvo un aumento en un día de un 24,78 % de detección de violaciones a los estándares arquitectónicos, dándole cumplimiento a la idea a defender planteada en la investigación.

Por lo anteriormente expuesto se concluye que con la herramienta para el tratamiento de violaciones de los estándares arquitectónicos del marco de trabajo Sauxe se disminuye el tiempo de reorganización del código y se mejora la detección de violaciones en el mismo lo cual constituye un valor agregado al marco de trabajo Sauxe.

## Recomendaciones

En función de mejorar la solución se recomienda:

- ✓ Extender la herramienta con funcionalidades que permitan detectar violaciones en los estándares arquitectónicos para la capa de presentación en el marco de trabajo Sauxe.

## Referencias bibliográficas

**Academia. 2015.** academia. *The downside of UML as a Complete Modeling Tool for Software Design.* 2015.

**Acuña, Karenny Brito. 2013.** eumed.net. *eumed.net.* [En línea] 2013. [Citado el: 3 de 5 de 2015.] http://www.eumed.net/libros/2009c/584/RUP%20Diseno%20e%20implementacion%20del%20sistema.html.

**AllPe. 2015.** ALLPE Ingeniería y Medio Ambiente. [En línea] 2015. [Citado el: 10 de 3 de 2015.] http://www.allpe.com/seccion_detalle.php?idseccion=27.

**Allstudies. 2015.** Allstudies.com: Estudios, Grados,Universidades. [En línea] 2015. [Citado el: 8 de 3 de 2015.] http://allstudies.com/ques-es-auditoria-contable.html.

**2014.** Apache. [En línea] 2014. [Citado el: 8 de 12 de 2014.] http://httpd.apache.org/docs/2.0/.

**Baryolo, Oiner Gómez. 2010.** *Solución informática de autorización en entornos multientidad y multisistema.* Universidad de las Ciencias Informáticas. La Habana: s.n., 2010. Tesis de Maestría.

**Buschmann, Frank, Henney, Kevlin, C.Schmidt, Douglas. 2007.** *Pattern-Oriented Software Architecture:On Patterns and Pattern Languages.* 2007. pág. 491. Vol. 5.

**Buschmann, Frank, y otros. 1996.** *Pattern – Oriented Software Architecture. A System of Patterns.* [ed.] John Wiley & Sons. 1996.

**Calderin, Leidisara Martínez. 2008.** Biblioteca Virtual de las Ciencias en Cuba. [En línea] 2008. [Citado el: 7 de 2 de 2015.] http://www.bibliociencias.cu/gsdl/collect/revistas/import/Control_interno.pdf.

**Camacho, Erika, Cardeso, Favio, Nuñez, Gabriel. 2004.** *Arquitectura de software. Guía de Estudio.* 2004.

**Carlos Reynoso, Nicolás Kiccillof. 2004.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires. 2004.

**Castro, Karolay Michell Coronel. 2012.** *Auditoría Informática orientada a los procesos críticos de crédito generados en la Cooperativa de Ahorro Y Crédito "Fortuna" aplicando el marco de trabajo COBIT.* ECC: Escuela Ciencias de la Computación. Ecuador: Universidad Técnica Particular de Loja, 2012. pág. 139, Tesis de Grado.

**Cervantes, Humberto. 2010.** SG Buzz. [En línea] 2010. http://sg.com.mx/revista/27/arquitectura-software.

**2001.** Checkstyle. *Checkstyle.* [En línea] 2001. [Citado el: 14 de 12 de 2014.] http://checkstyle.sourceforge.net/.

**Cuba, Proyecto ERP. 2008.** *Normas y estándares de Codificación del ERP.* Línea de Arquitectura, Universidad de Ciencias Informáticas. La Habana: s.n., 2008.

**Date, C.J. 2001.** *Introducción a los Sistemas de Bases de Datos.* 7ma Edición. S.l.: PEARSON EDUCACIÓN, México, 2001, 2001. ISBN: 968-444-419-2.

# Referencias bibliográficas

**Definición. 2008.** Definición_de. [En línea] 2008. http://definicion.de/lenguaje-de-programacion/.

**Developers, Google. 2012.** Google Developers. [En línea] 2012. [Citado el: 15 de 3 de 2015.] https://developers.google.com/java-dev-tools/codepro/doc/.

**Dictionary. 2015.** Dictionary.com. [En línea] 2015. http://dictionary.reference.com/browse/architectural.

**2015.** Doctrine. [En línea] 2015. [Citado el: 10 de 3 de 2015.] http://docs.doctrine-project.org/projects/doctrine1/en/latest/en/manual/introduction.html#what-isdoctrine.

**Doctrine. 2006.** Doctrine. *Doctrine Project.* [En línea] 2006. 14.] http://docs.doctrineproject.org/projects/doctrine1/en/latest/en/manual/introduction.html#what-is-doctrine.

**Fabri, F. 2015.** Softonic. [En línea] 2015. http://visualsvn-server.softonic.com/.

**Framework. 2015.** Zend Framework2. [En línea] 2015. [Citado el: 10 de 3 de 2015.] http://framework.zend.com/about.

**Ganesh, Gundai Sai. 2008.** *Requirements Engineering: Elicitation Techniques.* [ed.] Matemática y Ciencia de la Computación Departamento de Tecnología. S.l.: Universidad del Este, 2008.

**Garzás, Javier. 2012.** javiergarzas.com. [En línea] 1 de marzo de 2012. http://www.javiergarzas.com/2012/03/herramientas-de-calidad-software.html.

**—. 2011.** javiergarzas.com. [En línea] 2011. [Citado el: 17 de 3 de 2015.] http://www.javiergarzas.com/2011/01/duplicar-codigo.html.

**Gutierrez, J.J. y Escalona, M.J. 2008.** Asociación de Técnicos de Informática (ATI). [En línea] 2008. X Jornadas de Innovación y Calidad del . https://www.ati.es/IMG/pdf/EscalonaPapel.pdf.

**IEEE. 2012.** IEEE computer society. *The Community for Technology Leaders.* [En línea] 2012. http://cloudcomputing.ieee.org/csdl/proceedings/hase/2012/4912/00/4912a193.pdf.

**javaHispano, Asociación. 2004.** javaHispano. [En línea] 17 de mayo de 2004. [Citado el: 1 de 2 de 2015.] http://www.javahispano.org/antiguo_javahispano_org/2004/5/17/hammurapi-herramienta-libre-de-inspeccion-de-codigo-fuente-j.html.

**JavaScript. 2008.** The JavaScript Source. [En línea] 2008. [Citado el: 2 de 12 de 2014.] http://www.javascriptsource.com/.

**Kioskea. 2015.** Kioskea. [En línea] 2015. http://es.kioskea.net/contents/304-lenguajes-de-programacion.

**Knizhnik, Konstantin. 2002.** artho.com. [En línea] 27 de junio de 2002. http://www.artho.com/jlint/manual.html.

**Knowledge. 2015.** IT Knowledge portal. [En línea] 2015. http://www.itinfo.am/eng/software-development-methodologies/.

**Krathos. 2015.** KRATHOS:Investigación+Estrategia+Tecnología Gráfica. [En línea] 2015. [Citado el: 8 de 3 de 2015.] http://krathos.com.mx/?page_id=80.

# Referencias bibliográficas

**Kusek, Mario. 2001.** Bibliografía científica croata. [En línea] 2001. [Citado el: 5 de 2 de 2015.] https://bib.irb.hr/datoteka/85049.055_C28.pdf.

**Larman, Craig. 2003.** *UML y Patrones: Una introduccion al análisis y diseño orientado a objetos y al proceso unificado.* 2da Edición. 2003.

**—. 1999.** *UML y patrones:introduccion al análisis y diseño orientados a objetos.* Primera edición. 1999.

**Leahy, Paul. 2015.** Java Software Programming Tutorials and Resources. [En línea] 2015. [Citado el: 5 de 2 de 2015.] http://java.about.com/od/s/g/sourcecode.htm.

**López, José María. 2010.** RapidSVN. [En línea] 2010. [Citado el: 1 de 2 de 2015.] http://rapidsvn.softonic.com/.

**Martínez, Evelio. 2014.** Eveliux. [En línea] 2014. http://www.eveliux.com/mx/curso/estandares-y-organizaciones.html.

**Martínez, Rafael. 2013.** PostgreSQL-es. *Portal español sobre PostgreSQL.* [En línea] 2013. http://www.postgresql.org.es/sobre_postgresql.

**Mehdi Achour, Friedhelm Betz,Antony Dovgal y otros. 2014.** PHP. *PHP.* [En línea] 8 de 12 de 2014. http://php.net/manual/es/.

**Naiman, Gabriel. 2009.** 1, Buenos Aires: Grupo Pragma Consultores, Abril de 2009, Perspectiva.

**Netbeans. 2014.** Netbeans. [En línea] 2014. [Citado el: 8 de 12 de 2014.] https://netbeans.org/community/releases/80/.

**Paradigm. 2013.** Visual-Paradigm. *Sitio web.* [En línea] 2013. [Citado el: 9 de 12 de 2014.] http://www.visual-paradigm.com/aboutus/newsreleases/vpuml80.jsp.

*Patrón Modelo-Vista-Controlador.* **Romero, Yenisleidy Fernández y González, Yanette Díaz. 2012.** 1, Habana: s.n., enero-abril de 2012, Revista Digital de las Tecnologías de la Información y las Comunicaciones:Telem@tica, Vol. 11. ISSN 1729-3804.

**PgAdmin. 2011.** Puro Software. *Tu portal de software libre y gratuito.* [En línea] 2011. http://www.purosoftware.com/programacion-bases-de-datos/11-pg-admin-3.html.

**PMD. 2002.** PMD. *PMD.* [En línea] 2002. [Citado el: 13 de 12 de 2014.] http://pmd.sourceforge.net/.

**Pressman, Roger S. 2005.** *Software Engineering A Practitioner's Approach.* Séptima Edición. 2005.

**Rubira, Jorge. 2011.** GENBETA: dev desarrollo y software. [En línea] 10 de julio de 2011. http://www.genbetadev.com/herramientas/jslint-y-jshint-analizadores-de-codigo-javascript-online.

**Sánchez, Tamara Rodríguez. 2014.** *Metodología de desarrollo para la Actividad productiva de la UCI.* Universidad de Ciencias Informáticas. Ciudad Habana: s.n., 2014.

**Sencha. 2014.** What is Sencha Ext JS? *Sitio Web de Sencha.* [En línea] 2014. [Citado el: 5 de 12 de 2014.] http://www.sencha.com/products/extjs/.

# Referencias bibliográficas

**Sociales, Academia de Administración y. 2011.** Universidad Autónoma del Estado de Hidalgo. *Escuela Superior de Tlahuelilpan.* [En línea] Julio-Diciembre de 2011. [Citado el: 15 de 12 de 2014.] http://www.uaeh.edu.mx/docencia/P_Presentaciones/tlahuelilpan/sistemas/auditoria_informatica/auditoria_informatica.pdf.

**Software, Grupo de construcción de. 2010.** QualDev Group. [En línea] Universidad de los Andes,Bogotá Colombia, 25 de 6 de 2010. http://qualdev.uniandes.edu.co/wikiDev/doku.php?id=development:resources:tutorials:support:softwareevaluation:hammurapi.

**Sommerville, Ian. 2005.** *Ingeniería de Software.* Séptima Edición. Madrid: Pearson Educación, 2005.

**2008.** SonarQube. *SonarQube.* [En línea] 2008. [Citado el: 14 de 12 de 2014.] http://www.sonarqube.org/.

**Sperberg, Camilo. 2012.** unreal4u´s Personal Network. [En línea] 2012. [Citado el: 4 de 5 de 2015.] http://blog.unreal4u.com/2011/03/sobre-convenciones-ynotaciones-hungara-camelcase-etc/.

**Svensk, Magnus. 2012.** DiVA. [En línea] 2012. [Citado el: 4 de 5 de 2015.] http://urn.kb.se/resolve?urn=urn:nbn:se:hig:diva-12010.

**Tedechi, Nicolás. 2015.** ¿Qué es un Patrón de Diseño? *Microsoft Developer Network.* [En línea] 2015. [Citado el: 11 de 3 de 2015.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.

**Vilañez, Alexis. 2014.** *Herramienta de calidad software Check Style.* Universidad católica del Ecuador. Sede Ibarra. 2014. [Citado el: 4 de 5 de 2015] https://prezi.com/eiabhb_dhkrf/herramientas-de-calidad-software-check-style/

**Visconti, Marcello. 2012.** *Fundamentos de Ingeniería de Software.* s.l. : Universidad Técnica de Federico Santa María, 2012.

**WebSecurity. 2007.** WebSecurity.es. *WebSecurity.es.* [En línea] 2007. [Citado el: 12 de 12 de 2014.] http://www.websecurity.es/herramientas-realizar-auditorias-nuestro-programas.

**Zend-Technologies. 2006.** Zend Framework. [En línea] 2006. [Citado el: 8 de 12 de 2014.] http://framework.zend.com/about/.

## Anexos

**Anexo 1. Instrumento de evaluación de la métrica TOC.**

| Clase | Cantidad de procedimientos | Responsabilidad | Complejidad | Reutilización |
|---|---|---|---|---|
| GestnomController | 13 | Alta | Alta | Baja |
| gestmodulo | 4 | Baja | Baja | Alta |
| NomTipoViolacionModel | 5 | Baja | Baja | Alta |
| NomViolacionPatronesModel | 8 | Media | Media | Alta |
| NomTipoViolacion | 4 | Baja | Baja | Alta |
| NomViolacionPatrones | 6 | Media | Media | Alta |
| BaseNomTipoViolacion | 1 | Baja | Baja | Baja |
| BaseNomViolacionPatrones | 1 | Baja | Baja | Alta |

**Anexo 2. Instrumento de evaluación de la métrica RC.**

| Clase | Cantidad de relaciones | Acoplamiento | Complejidad | Reutilización | Cantidad de pruebas |
|---|---|---|---|---|---|
| GestnomController | 4 | Alta | Alta | Baja | Alta |
| gestmodulo | 1 | Baja | Baja | Alta | Baja |
| NomTipoViolacionModel | 1 | Baja | Baja | Alta | Baja |
| NomViolacionPatrones Model | 1 | Baja | Baja | Alta | Baja |
| NomTipoViolacion | 1 | Baja | Baja | Alta | Baja |
| NomViolacionPatrones | 1 | Baja | Baja | Alta | Baja |
| BaseNomTipoViolacion | 0 | Ninguno | Baja | Alta | Baja |
| BaseNomViolacionPatrones | 0 | Ninguno | Baja | Alta | Baja |

**Anexo 3. Cuestionario aplicado para realizar el pre-experimento.**

## Cuestionario para validar la idea a defender de la herramienta para el tratamiento de violaciones a los estándares arquitectónicos en el marco de trabajo Sauxe

En la actualidad el Departamento de Desarrollo de Componente para la creación de aplicaciones de gestión utiliza el marco de trabajo Sauxe el cual brinda una base tecnológica para la creación de las mismas. Antes que las aplicaciones sean liberadas se comprueba que los componentes desarrollados cumplan con los estándares arquitectónicos definidos, de no ser así se procede a su estabilización, la cual es realizada de forma manual. Teniendo en cuenta el planteamiento anterior se decide desarrollar una herramienta que sea capaz de disminuir el tiempo de reorganización y aumentar la detección de violaciones a los estándares arquitectónicos definidos en el marco de trabajo Sauxe.

En el marco de esta investigación se entiende por violación a los estándares arquitectónicos al incumplimiento de las reglas, normas o características que permitan asegurar el diseño y la implementación de un software. Por reorganización de código fuente se entiende como forma de estructurar nuevamente el código fuente atendiendo a las reglas definidas previamente.

Para validar los aspectos antes expuestos, se elaboró el presente cuestionario para evaluar el proceso en los dos ambientes, al realizar la estabilización de forma manual y con la herramienta. El mismo será respondido por la **Ing**. Katia Saria Preval, el **Ing**. Inoelkis Velázquez Osorio y el **Ing**. René R. Bauta Camejo.

Para llevar a cabo este proceso se escogieron teniendo en cuenta la complejidad de las clases se seleccionaron las siguientes:

- **Complejidad Alta**: GestSistemaController (con su model y domain). Estas clases responden a la funcionalidad Gestionar sistema del sistema de seguridad Acaxia. Para configurar la seguridad en un nuevo componente es necesario registrarlo mediante esta funcionalidad, así como configurar la conexión a base de datos. La complejidad de implementación es alta por el alto grado de responsabilidad que presenta esta funcionalidad. La misma cuenta con 1606 sentencias y 52 funcionalidades.

- **Complejidad Media**: GestFuncionalidadController (con su model y domain). Estas clases responden a la funcionalidad Gestionar funcionalidad del sistema

Acaxia. Luego de registrado el sistema se procede a registrar las funcionalidades con que cuenta. Su complejidad en la implementación no es crítica como la funcionalidad anterior pero presenta un número considerable de funciones y sentencias. Esta clase presenta 175 sentencias con 7 funcionalidades.

**Complejidad Baja:** GestNomIdiomaController (con su model y domain). Estas clases responden a la funcionalidad Gestionar idioma del sistema Acaxia. Al definir un usuario uno de los campos a especificar es el idioma a utilizar cuando inicie en el sistema. Esta funcionalidad presenta complejidad baja debido a que no representa dificultad en su implementación. La misma cuenta con 87 sentencias y 6 funcionalidades.

**Antes del desarrollo de la herramienta para el tratamiento de violaciones a los estándares arquitectónicos:**

1. **Atendiendo a la complejidad de las siguientes clases, en un día diga qué cantidad de violaciones usted fue capaz de detectar en el módulo Gestionar sistema.**

   3 GestNomIdiomaController, NomIdiomaModel y NomIdioma.

   19 GestFuncionalidadController, DatFuncionalidadModel y DatFuncionalidad.

   90 GestSistemaController, DatSistemaModel y DatSistema.

2. **Atendiendo a la complejidad de las clases, en un día diga cuanto tiempo demora usted en detectar las violaciones en cada una de ellas**

15 min GestNomIdiomaController, NomIdiomaModel y NomIdioma.

25 min GestFuncionalidadController, DatFuncionalidadModel y DatFuncionalidad.

4/0 min GestSistemaController, DatSistemaModel y DatSistema.

3. **Atendiendo a la complejidad de las clases, en un día diga qué tiempo le conlleva la reorganización del código fuente.**

30 min GestNomIdiomaController, NomIdiomaModel y NomIdioma.

5 horas GestFuncionalidadController, DatFuncionalidadModel y DatFuncionalidad.

1 día GestSistemaController, DatSistemaModel y DatSistema.

**Después del desarrollo de la herramienta para el tratamiento de violaciones a los estándares arquitectónicos:**

1. **Atendiendo a la complejidad de las clases, en un día diga qué cantidad de violaciones fueron detectadas.**

   5 GestNomIdiomaController con su respectiva model y domain.

25 GestFuncionalidadController con su respectiva model y domain.

115 GestSistemaControlller con su respectiva model y domain.

2. **¿Cuánto tiempo demoró en detectar las violaciones en cada una de las clases?**

1 Seg GestNomIdiomaController con su respectiva model y domain.

2 Seg GestFuncionalidadController con su respectiva model y domain.

3 Seg GestSistemaController con su respectiva model y domain.

3. **¿Qué tiempo requirió la reorganización del código fuente?**

15 Min GestNomIdiomaController con su respectiva model y domain.

2 HORAS GestFuncionalidadController con su respectiva model y domain.

5 HORAS GestSistemaController con su respectiva model y domain.

4. **¿Cree usted factible la utilización de la herramienta? ¿Por qué?**

Sí, ya que el proceso de estabilización del código en el marco de trabajo Sauxe puede provocar afectaciones en el plan de trabajo. y con la herramienta se disminuye el tiempo empleado, así como detectar más violaciones que a simple vista no fueron detectadas.

Ing. Katia Saria Preval

**Anexo 4: Acta de aceptación de la herramienta.**

UCI
Universidad de las Ciencias Informaticas

CENTRO DE INFORMATIZACIÓN DE ENTIDADES
DEPARTAMENTO DE DESARROLLO DE COMPONENTES

09 de junio de 2015

*A quien pueda interesar:*

Por este medio se hace constar que la solución ***Herramienta para el tratamiento de violaciones a los estándares arquitectónicos del marco de trabajo Sauxe*** de los autores ***Alicia Chacón Rodríguez y Jorge Burgos Díaz*** fue sometida a una revisión técnica en la cual se detectaron 7 no conformidades que fueron resueltas quedando esta solución estable y lista para su posterior uso.

Como parte del desarrollo de la solución se elaboraron y entregaron los siguientes artefactos:

1. Modelo conceptual.
2. Descripción de requisitos (10).
3. Modelo de diseño (1).
4. Diseños de casos de prueba (10).
5. Modelo de datos
6. Archivos * vpp generados con el Visual Paradigm
7. Código fuente

(https://ceige-repo.uci.cu/sauxe/Sauxe/Raiz/Branches/Sauxe_v2.3_Tools/)

Para que así conste firman a continuación los miembros del equipo que realizo la revisión, el autor y los tutores del trabajo.

Dado a los 09 días del mes de junio de 2015.

| Nombre y apellidos | Firma |
|---|---|
| **Revisores:**<br>Ing. Inoelkis Velazquez Osorio | |
| **Autor (es):**<br>Alicia Chacón Rodríguez<br>Jorge Burgos Díaz | |
| **Tutores:**<br>Ing. René R. Bauta Camejo<br>Ing. Katia Saria Preval | |

**J' Departamento de Desarrollo de Componentes**
**René R. Bauta Camejo**

## Glosario de términos

**A.**

**Artefactos:** productos tangibles del proyecto que son creados, modificados y usados dentro de las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

**C.**

**Componente:** unidad de composición de aplicaciones de software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio.

**E.**

**Entidad:** forma parte de la estructura de un país y puede comportarse como un ministerio, empresa, asociación, área o cargo.

**Estándar:** que sirve de modelo o punto de referencia para medir o valorar cosas de la misma especie.

**H.**

**Herramienta:** es un objeto elaborado a fin de facilitar la realización de una tarea. Sirve como recurso.

**M.**

**Métrica:** medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

**R.**

**Reusabilidad:** capacidad de un componente y un subsistema para ser usado por otras aplicaciones en otros escenarios. Esto minimiza la duplicación de componentes así como el tiempo de implementación.

**S.**

**Sistema:** es un conjunto organizado de objetos o partes interactuantes e interdependientes, que se relacionan formando un todo unitario y complejo.

**Software:** conjunto de instrucciones que los ordenadores emplean para manipular datos.

**V.**

**Validación:** confirmación mediante el suministro de evidencia objetiva de que se han cumplido los requisitos para una utilización o aplicación específica prevista.