Universidad de las Ciencias Informáticas

Facultad 3

Trabajo de Diploma para optar por el título de Ingeniero en

Ciencias Informáticas

Diseño e implementación de los procesos de violaciones y responsables para el módulo Verificación Fiscal del Sistema de informatización de la Gestión de las Fiscalías fase II.

Autor(es):

Kenia López Hernández

Nilson Mulet Morales

Tutor:

Ing. Felipe Hernández Salazar

La Habana, Junio de 2014.


"Si la juventud
falla, todo fallará"

DECLARACIÓN DE AUTORÍA

Se declara que Kenia López Hernández y Nilson Mulet Morales son los únicos autores del trabajo de diploma Diseño e implementación de los procesos de violaciones y responsables para el módulo Verificación Fiscal del Sistema de Informatización de la Gestión de las Fiscalías fase II y se le reconocen a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ____ días del mes de _____ del año 2014.

_____ _____

Firma del autor

Kenia López Hernández

Firma del autor

Nilson Mulet Morales

_____

Firma del Tutor

Ing. Felipe Hernández Salazar

DATOS DE CONTACTO

Datos del tutor:

Nombre: Ing. Felipe Hernández Salazar

Ingeniero en Ciencias Informáticas. Labora en el proyecto SIGEFII, ocupando el rol de programador.

Correo electrónico: fsalazar@uci.cu

Datos del Autor

Nombre: Kenia López Hernández

Correo electrónico: klopez@estudiantes.uci.cu

Datos del Autor

Nombre: Nilson Mulet Morales

Correo electrónico: nmulet@estudiantes.uci.cu

A mis padres, por su apoyo incondicional y ser un ejemplo de esfuerzo y dedicación en mi vida. A mi mamá que siempre ha creído en mí en todos los momentos difíciles de mi carrera. A mi papá que aunque este lejos de mí, siempre se ha sacrificado mucho por mí para lograr este resultado.

A Michael por ayudarme a crecer en este año tan difícil de mi carrera y por enseñarme a levantarme y no dejarme caer.

Kenia López Hernández

A mis dos familias por apoyarme siempre en todo momento, aun cuando muchas veces no sabía cómo explicar que no tenía mucho tiempo para dedicarles. A mi mamá, lo más grande del mundo para mí por saber entenderme y haber sido siempre tan paciente, gracias mi vieja. A mi segunda mamá por haber asumido la titánica tarea de soportarme y entenderme siempre. A mi papá por sus consejos, por ser amigo, espero que este título compense ese tiempo que nunca ha sido suficiente. A mi segundo padre, siempre te voy a estar agradecido por haberme acogido como un hijo propio. A mis abuelos, especialmente a mi abuela Bárbara y a Juana María. A mis tíos por haber sido siempre mis amigos para todo. A mis hermanos que quiero con la vida.

A mi niña linda por estar a mi lado apoyándome en todo momento.

Nilson Mulet Morales

Agradecimientos

A nuestros familiares por su apoyo incondicional. A nuestros compañeros de grupo que vienen con nosotros desde primer año y a los que se integraron en años posteriores. A los compañeros del proyecto SIGEF fase II por su paciencia, apoyo y ayuda en los momentos difíciles. Agradecimientos especiales a Lázaro por ser incondicional en todo momento. A nuestro tutor por apoyarnos .Gracias a todos los que de una forma u otra nos ayudaron con el desarrollo de este trabajo.

Kenia López Hernández y Nilson Mulet Morales.

RESUMEN

La Fiscalía General de la República de Cuba es el órgano del estado que tiene como objetivos fundamentales velar por el control y el cumplimiento de las leyes en la sociedad. En este órgano se encuentra el área Verificación Fiscal, que se encarga de velar por el cumplimiento de la legalidad y combatir las manifestaciones de corrupción. Dentro de esta área se gestionan las violaciones de la legalidad y los responsables que se generan de las verificaciones fiscales que se realizan en las entidades.

El presente trabajo de diploma tiene como propósito el diseño y la implementación de los procesos detección de violaciones y responsables. Para el cumplimiento de este propósito se realiza una descripción de la metodología de desarrollo, las herramientas y los lenguajes empleados en el desarrollo del subsistema. Se presentan además los artefactos generados en las etapas de diseño, implementación y pruebas, sobre la base del análisis de la arquitectura que soporta el sistema. Al finalizar se muestran los resultados obtenidos después de haber aplicado las pruebas pertinentes para detectar y corregir los errores antes de la entrega al cliente.

El desarrollo de esta propuesta proveerá a las fiscalías cubanas de una solución informática que contribuirá al control de la gestión de la información que se tramita y a la celeridad en la ejecución de los procesos. Se espera que los fiscales tengan a su disposición un sistema que les permita mejorar la realización de los procesos.

Palabras Claves:

Verificación Fiscal,diseño,implementación,violaciones,responsables,pruebas.

INTRODUCCIÓN

El acelerado desarrollo de las tecnologías de la informática y las comunicaciones (TIC) ha traído consigo cambios en la sociedad. Dichas tecnologías han contribuido a que las personas modifiquen sus hábitos, patrones de conducta, forma de pensar y actuar. (1)

Cuba ha declarado su propia política para la informatización. Esta política tiene su base en la utilización ordenada y masiva de estas tecnologías para satisfacer las necesidades de información y conocimiento de todas las personas. (2) La Universidad de las Ciencias informáticas (UCI) se funda con el fin de servir de soporte a la industria cubana de la informática mediante la producción de aplicaciones y servicios que faciliten la informatización de la sociedad. En ella se encuentra el Centro de Gobierno Electrónico (CEGEL), el cual está situado en la facultad 3. El mismo tiene la misión de satisfacer necesidades de clientes gubernamentales mediante el desarrollo de productos y servicios de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia, a partir de un personal altamente calificado. La UCI cuenta con numerosos proyectos productivos, los cuales tienen la tarea de informatizar procesos claves en determinadas instituciones. Uno de estos proyectos lo compone el Sistema de Informatización de la Gestión de las Fiscalías fase II (SIGEF II), encargado de informatizar los procesos de la Fiscalía General de la República (FGR). La FGR se divide en dos instancias: municipales y provinciales. Estas fiscalías representan el órgano de dirección del trabajo fiscal en su territorio y su máxima autoridad es el fiscal jefe provincial o municipal correspondiente.

La FGR esta compuesta por cuatro áreas, una de estas es Verificación Fiscal (VF). La misma tiene como objetivos:

- ✓ Comprobar el cumplimiento de las disposiciones referidas al objeto social; el uso, destino y preservación de los recursos materiales y financieros; velando porque se utilicen de acuerdo a los fines para los que fueron destinados o producidos; así como los sistemas de control establecidos.
- ✓ Velar por el uso del capital social del Estado en las entidades jurídicamente privadas en las que tenga participación.
- ✓ Prevenir, descubrir y enfrentar las manifestaciones de corrupción administrativa.
- ✓ Contribuir a fortalecer la disciplina estatal, determinar la responsabilidad laboral, material o penal de los infractores de la Ley, incluidos los colaterales en los casos que así se compruebe.
- ✓ Comprobar y exigir la erradicación de las violaciones detectadas.

- ✓ Realizar propuestas de normativas o modificaciones a las vigentes que resulten omisas, o no se correspondan con las necesidades actuales. (3)

Actualmente dos de los procesos que se llevan a cabo en VF lo constituye detección de violaciones y responsables de la legalidad donde se emiten tanto las violaciones en las que incurre la entidad objeto de una verificación como los responsables de las mismas, facilitando de esta manera la radicación de las violaciones detectadas, las causas y condiciones que la generaron así como la solicitud de medidas a los responsables.

La FGR tiene como base en sus aplicaciones el Sistema de Informatización de la Gestión de la Fiscalía fase I, el cual no incluye el área VF. Por esta razón, no se encuentran informatizados los procesos detección de violaciones y responsables, además, los procedimientos para ejecutar los procesos por instancia no se realizan de forma homogénea. Estos procesos generan una alta cantidad de información a procesar, traducida en un gran volumen de documentos a desarrollar, lo que afecta el tiempo y esfuerzo de los fiscales. La planificación de las actividades se realiza de manera manual, provocando atrasos e incumplimientos, lo que disminuye la calidad de las actividades previstas a desarrollar en los procesos. La documentación y notificaciones de cada instancia se envían por correo electrónico o postal, y los documentos que no mantienen relevancia, pasados los cinco años de permanencia en la fiscalía, se destruyen. Esta forma de trabajar trae consigo que las consultas a los documentos sean complejas y se dificulte su acceso futuro, lo cual afecta la disponibilidad de la información.Todo esto ocasiona que el acceso a la información sea difícil para los fiscales por lo que se hace complejo controlar y supervisar por parte de todas las instancias la obtención de los reportes estadísticos sobre sus procesos.

Partiendo de la situación anteriormente descrita, el equipo de análisis del proyecto documentó todos los procesos de negocio que se realizan en las fiscalías y se identificaron las principales funcionalidades a automatizar según las necesidades del cliente. Posteriormente se identificaron un conjunto de requisitos funcionales y no funcionales que debe cumplir el sistema que se necesita desarrollar para la FGR. Luego de haber realizado el modelado de los procesos del negocio y el levantamiento y especificación de requisitos, se diseñaron los prototipos de interfaz de usuario mediante talleres realizados internamente en el equipo de desarrollo y luego se validaron estos prototipos en talleres de validación con el cliente.

A continuación se presenta un resumen de la información existente en cuanto a requisitos funcionales (RF) y requisitos no funcionales (RNF).

Procesos	RF	RNF
Proceso detección de violaciones	36	19
Proceso detección de responsables	18	

Tabla 1: Información existente

Con la información anteriormente descrita se decide entonces continuar con el diseño y la implementación de los procesos de detección de violaciones y responsables del SIGEF II.

A partir de la situación problemática identificada se plantea el siguiente **problema a resolver:** ¿Cómo mejorar la gestión de los procesos de violaciones y responsables del área Verificación Fiscal de la Fiscalía General de la República, de forma tal que se contribuya al control y a la celeridad en la ejecución de los mismos?

En función de resolver el problema propuesto se define como **objetivo general:** Desarrollar los requisitos de software correspondiente a los procesos de violaciones y responsable del SIGEF II de manera que se mejore la gestión del proceso.

El **objeto de estudio** es el Proceso de desarrollo de software de gestión en la informática jurídica y el **campo de acción** el diseño, implementación y validación de los procesos violaciones y responsables del área VF. Se define como **idea a defender:** El diseño, implementación y validación de los procesos de violaciones y responsables del SIGEF II permitirán que se mejore la gestión del proceso contribuyendo al control y a la celeridad en la ejecución del mismo.

Para darle cumplimiento al objetivo general, se trazaron los siguientes **objetivos específicos:**

- Diseñar los procesos de violaciones y responsables del módulo Verificación Fiscal del proyecto SIGEF II, teniendo en cuenta la metodología de desarrollo, los requisitos definidos y la arquitectura de software del proyecto.
- Implementar los procesos de violaciones y responsables del módulo Verificación Fiscal del proyecto SIGEF II, mediante las herramientas y lenguajes definidos en el proyecto para la obtención del producto.
- Validar la solución propuesta mediante pruebas unitarias y del sistema.

Para cumplir los objetivos específicos antes expuestos se realizarán a lo largo de todo el proceso investigativo las siguientes tareas de la investigación:

- Análisis de los procesos que se realizan en la detección de violaciones en una verificación e investigación fiscal así como de los responsables involucrados en estas.
- Estudio de la metodología de desarrollo del proyecto productivo SIGEF II.
- Estudio de los requisitos definidos por el proyecto productivo SIGEF II.
- Estudio de la arquitectura del proyecto productivo SIGEF II.
- Elaboración de los diagramas de clases y de secuencia del diseño para los procesos asociados a las violaciones y responsables del módulo Verificación Fiscal del proyecto del SIGEF II.
- Validación del diseño a través de métricas asociado a los procesos de violaciones y responsables.
- Implementación de las funcionalidades de los procesos de violaciones y responsables del módulo Verificación Fiscal del proyecto del SIGEF II.
- Validación de los resultados obtenidos a través de las pruebas de caja negra.
- Validación de los resultados obtenidos a través de las pruebas de caja blanca.
- Validación de la propuesta de solución de la investigación frente al problema planteado.

Para la realización de la investigación se aplicaron los métodos científicos siguientes:

Métodos teóricos:

- **Analítico-Sintético:** se emplea para el análisis de toda la información referente a la investigación, y así llegar a obtener conclusiones necesarias del trabajo.
- **Modelación:** utilizado para la elaboración de los diferentes diagramas que ayudaron a una mejor visión de la lógica del proceso en la FGR, así como de las funcionalidades que debe cumplir el sistema.

Estructura del documento:

El presente trabajo de diploma está compuesto por tres capítulos estructurados de la siguiente manera:

Capítulo 1. Fundamentación teórica. En el capítulo 1 se realiza una descripción de las herramientas, metodologías y lenguajes usados en el proyecto productivo SIGEF II para poder darle solución al problema planteado. También se enuncian los principales conceptos sobre el negocio que ayudaron al desarrollo del problema planteado.

Capítulo 2. Diseño e implementación. El capítulo 2 describe el diseño realizado que incluye los diagramas de clases necesarios haciendo uso de los patrones de diseño.

Contiene la validación del diseño mediante métricas propuestas por Pressman. Además abarca el desarrollo de todo el trabajo correspondiente con la implementación del sistema en términos de componentes.

Capítulo 3. Validación de la propuesta de solución. El capítulo 3 muestra las pruebas realizadas a la solución y sus resultados para asegurar el correcto funcionamiento de la aplicación.

CAPITULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En el presente capítulo se enuncian los conceptos fundamentales, principalmente del negocio, permitiendo un mejor entendimiento de los procesos pertenecientes al área Verificación Fiscal. Se describe la arquitectura usada en el proyecto productivo SIGEF II así como los patrones y métricas a utilizar.

1.2. Conceptos fundamentales

Para facilitar la comprensión del proceso VF fue preciso el estudio de algunos términos o conceptos empleados en instituciones fiscales y que son muy importantes en el desarrollo de las actividades que son realizadas por los fiscales. Estos son algunos de los más utilizados:

1.2.1. Fiscalía General de la República

Es el órgano del Estado al que corresponde, como objetivos fundamentales, el control y la preservación de la legalidad, sobre la base de la vigilancia del estricto cumplimiento de la Constitución, las leyes y demás disposiciones legales, por los organismos del Estado, entidades económicas y sociales y por los ciudadanos; y la promoción y el ejercicio de la acción penal pública en representación del Estado. (3)

1.2.2. Fiscalía Provincial

Es el órgano de dirección del trabajo fiscal en su territorio, a la cual corresponde además el establecimiento y desarrollo de la colaboración con otras entidades. Está integrada por el fiscal jefe provincial, quien es su máxima autoridad, los vice fiscales jefes provinciales y por varios departamentos (4)

1.2.3. Fiscalía Municipal

Es el órgano de dirección del trabajo fiscal en su territorio, a la cual corresponde además el establecimiento y desarrollo de la colaboración con otras entidades. Está integrada por el fiscal jefe municipal, quien es su máxima autoridad y en su caso por el vice-fiscal jefe municipal, los fiscales y el personal administrativo. (4)

1.2.4. Legalidad o primacía de la ley

Es un principio fundamental conforme al cual todo ejercicio del poder público debería estar sometido a la voluntad de la ley y de su jurisdicción y no a la voluntad de las personas. Por esta razón se dice que el principio de legalidad establece la seguridad jurídica. (5)

1.3. Metodología utilizada en el proyecto SIGEF II

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. (6) Tiene como principal objetivo aumentar la calidad del software que se produce en todas y cada una de sus fases de desarrollo. (7)

Desde la fase I, el proyecto SIGEF viene rigiéndose por la metodología RUP (Proceso unificado de desarrollo por sus siglas en inglés Rational Unifield Proccess), principalmente por la ventaja de guiar a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado, mientras se balancean los requisitos y los riesgos del proyecto. El mismo de manera clara describe los diversos pasos involucrados en la captura de los requisitos, detalla qué entregables producir y cómo desarrollarlos (8). Actualmente en la universidad se está llevando a cabo un ciclo de vida definido como parte del Programa de Mejora (PM), el cual posee fases e iteraciones que sugieren qué hacer, no cómo hacerlo. El uso de esta guía de trabajo permitirá manejar correctamente proyectos a largo plazo, haciendo especial énfasis en la generación de artefactos bien documentados que facilitan la capacitación y transferencia del producto. (9)

Rational Unified Process (RUP).

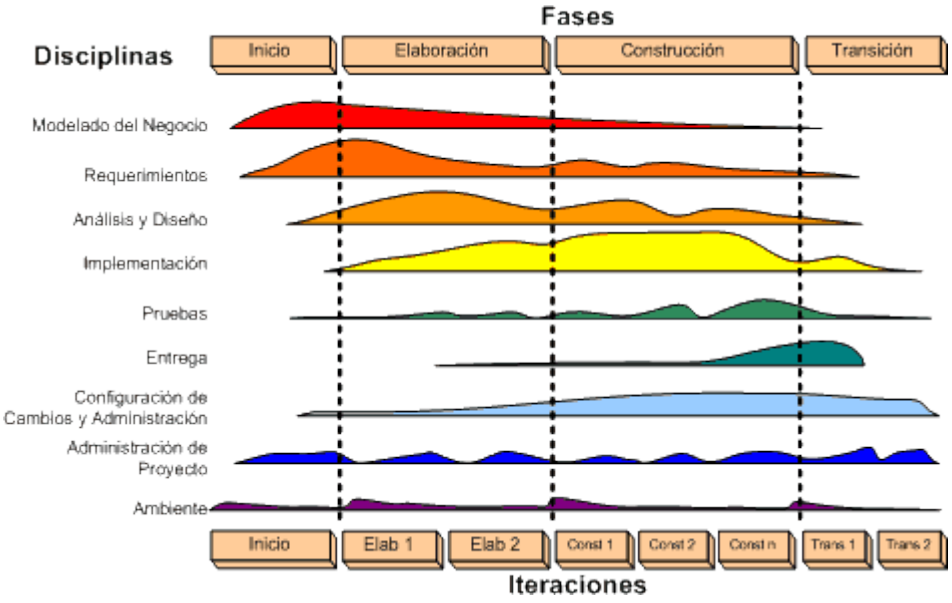


Figura 1: Fases y disciplinas de RUP.

La metodología RUP “Divide en 4 fases el desarrollo del software”: Inicio, donde se describe el negocio y se identifican los casos de uso. Elaboración, aquí se define la arquitectura del sistema y se obtiene una aplicación ejecutable. Construcción, donde se obtiene un producto listo para su utilización y Transición donde finalmente se instala el

software en condiciones reales. En cada ciclo se produce una nueva versión del sistema y cada versión es un producto preparado para su entrega. El producto terminado satisface todas las necesidades de los usuarios" (10).

Programa de mejora

Programa de Mejora que define el Centro de Calidad para Soluciones Informáticas (CALISOFT). Está basado en el modelo CMMI (Integración de Modelos de Madurez de Capacidades), teniendo como el objetivo principal de la UCI lograr una certificación internacional del nivel 2 de este modelo. El mismo preverá un crecimiento en cuanto a capacidades y madurez que se enfoca en todos los procesos alcanzando una mejora considerable en el ciclo de vida dentro del desarrollo de software, mayor productividad, presupuestos predecibles en los proyectos productivos y mayor calidad en los productos (11).

Aspectos importantes de CMMI

¿Qué es CMMI?

CMMI constituye una vista integradora de mejora de procesos a través de múltiples disciplinas y provee una orientación para la calidad de procesos. Cuenta con dos enfoques para la mejora del proceso: la capacidad del proceso y la madurez de la organización. (12) Los modelos de CMMI soportan cada enfoque con una representación. La capacidad del proceso tiene una representación continua para una sola área o conjunto de áreas de proceso mientras que la madurez de la organización según el PM está dada por una representación escalonada compuesta por 5 niveles de madurez para un conjunto definido de áreas de proceso de la organización. (12)

Ciclo de vida del PM

La realización de este trabajo está basada en el ciclo de vida de proyectos incluido en el PM que define CALISOFT y en un modelo de desarrollo iterativo e incremental constituido por cuatro fases: Inicio, Elaboración, Construcción y Transición. Según este ciclo de vida, el proceso de desarrollo incluye las siguientes disciplinas: Estudio Preliminar, Modelado del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas Internas, Pruebas de Liberación, Despliegue y Soporte. (11)

Después de realizado el estudio previo correspondiente a la fase de inicio incluida en este ciclo de vida de proyecto, se decide adoptar un modelo guiado por el PM con elementos de RUP para regir el proceso de desarrollo de software de la segunda fase del proyecto.

Para este trabajo servirán de guía las siguientes fases e iteraciones:

Fase de Elaboración

Requisitos: Los artefactos obtenidos en esta fase son: Especificación de Requisitos de Software.

Análisis y Diseño: Se obtiene el Modelo de Diseño como artefacto correspondiente a esta disciplina el cual incluye los Diagramas de clases del diseño, Diagrama de Paquetes y Descripción de las clases.

Fase de Construcción

Implementación y Pruebas: Se realiza la implementación del sistema y las pruebas internas al software para garantizar su correcto funcionamiento. En esta fase los artefactos generados son: Código fuente del software, Script de la base de datos, Diseño de Casos de Pruebas basado en Requisitos y Estándares de codificación para PHP.

1.4. Arquitectura del proyecto SIGEF II.

La arquitectura utilizada en el proyecto SIGEF II, tiene como objetivo estandarizar y agilizar la construcción del sistema. Está basada en componentes, incrementando el grado de reutilización de la misma, además de ser portable, fácil de mantener, disponible, modificable, escalable y segura. Esta arquitectura agiliza el desarrollo haciendo uso de patrones, permitiendo el desarrollo del trabajo en paralelo. (13) Se sustenta sobre una serie de componentes orientados en dos perspectivas: horizontal (ver figura 1) y vertical (ver figura 2) que comprenden, la organización de las partes del sistema para el cumplimiento de los requisitos funcionales en módulos y algunos de los requisitos no funcionales en niveles. A continuación se muestra la incorporación del módulo VF en el proyecto SIGEF II.

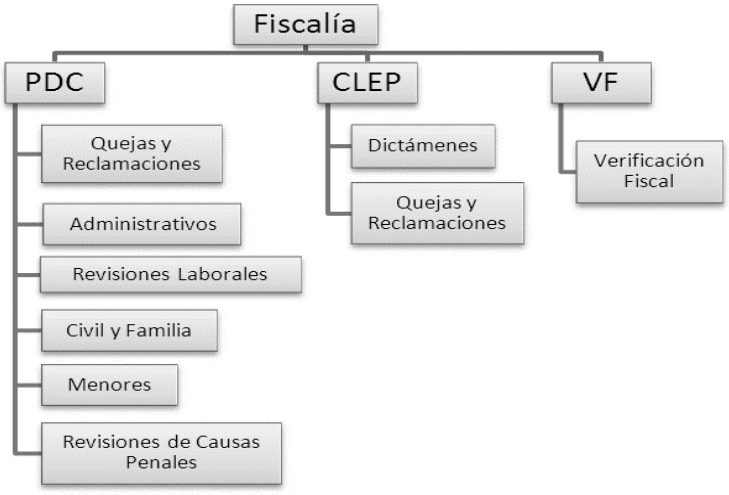


Figura 2: Representación Horizontal de la Arquitectura

La arquitectura del módulo VF según la representación horizontal se encuentra dentro del subsistema VF de SIGEF II (ver figura 2). Dentro de dicho módulo se encuentran los procesos de detección de violaciones y responsables los cuales serán objeto de estudio de la presente investigación.

La arquitectura del módulo VF está basada en el patrón arquitectónico modelo vista controlador (MVC). Se le agrega una estructura organizativa en la capa de modelo en la cual se agrupan las clases encargadas de encapsular la lógica de negocio y recibe el nombre de Negocio.

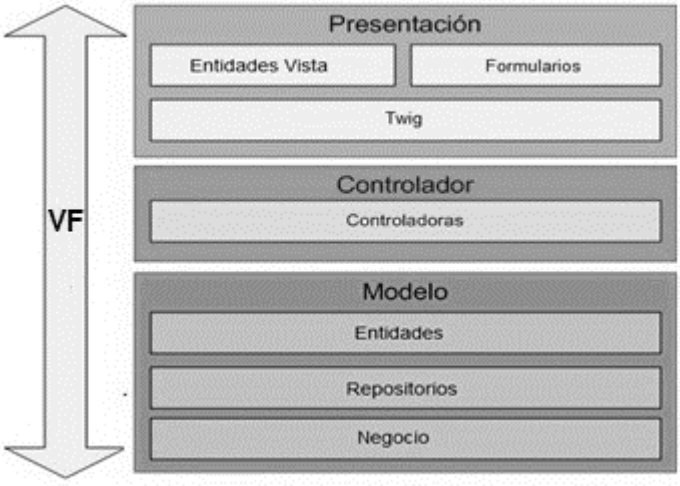


Figura 3: Representación vertical de la arquitectura de cada módulo de SIGEF II.

La Figura 3 describe la representación vertical de cada módulo de la aplicación donde el nivel superior encapsula las interfaces de usuario, entidades de presentación y formularios necesarios para la interacción con el cliente. La capa controlador contiene las clases controladoras que responden a una acción solicitada por el usuario. Esta se encarga de recibir una petición, procesar la información, hacer un pedido al modelo y devolver una respuesta a los controladores los cuales a su vez la envían a la vista. La capa de modelo es la capa inferior y su función es el manejo de los datos para visualizarlos o procesarlos. (13).

Como parte de la propuesta de la arquitectura del SIGEF II se incorpora un subsistema denominado arquitectura base, que sirve de apoyo en el desarrollo del módulo VF, reduciendo la implementación a los requisitos funcionales básicos, y permitiendo que el código sea altamente reutilizable, limitando sus errores y vulnerabilidades (ver figura 3).

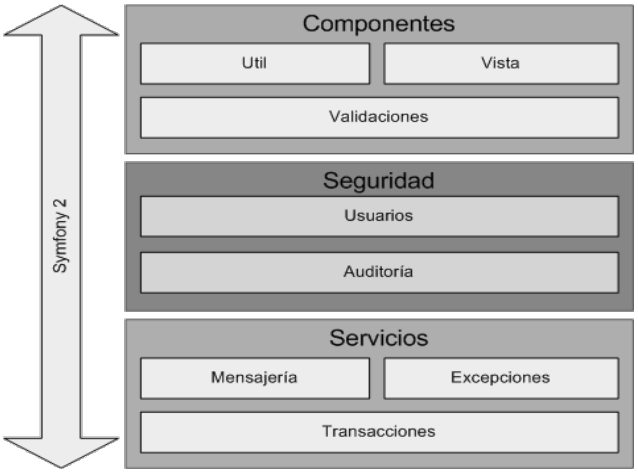


Figura 4: Subsistema Arquitectura Base.

Este subsistema está formado por componentes donde su primer nivel es precisamente una capa con ese nombre, la que encapsula una serie de funcionalidades de uso común en toda la aplicación relacionadas con la capa de presentación de cada módulo, además de configuraciones para el uso de variables globales y validaciones. La capa de seguridad se encarga de la gestión de usuarios, mostrar información de las trazas de usuarios y aspectos importantes como el control de acceso a cada objeto de la aplicación. El siguiente nivel se encarga de garantizar el cumplimiento de conceptos paralelos a todo el sistema como la creación de servicios, mensajería, tratamiento de excepciones y manejo transaccional garantizando la integridad de los datos. (14)

1.4.1. Patrón de arquitectura

Los patrones arquitectónicos son una descripción de un problema particular y recurrente de diseño, que aparece en contextos específicos y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí. (15)

Patrón Modelo-Vista-Controlador (MVC)

Este patrón es utilizado ya que el marco de trabajo empleado Symfony, en su versión 2, permite implementarlo a través de una estructura bien definida. (Ver figura 6). Dicha estructura permite que la arquitectura MVC separe la lógica de negocio (el modelo) y la presentación (la vista), lo que posibilita un mantenimiento más sencillo de las aplicaciones.

El modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio. La vista transforma el modelo en una página web que permite al usuario interactuar con ella. El controlador es el encargado de aislar al modelo y a la vista, además

procesa las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. (16)

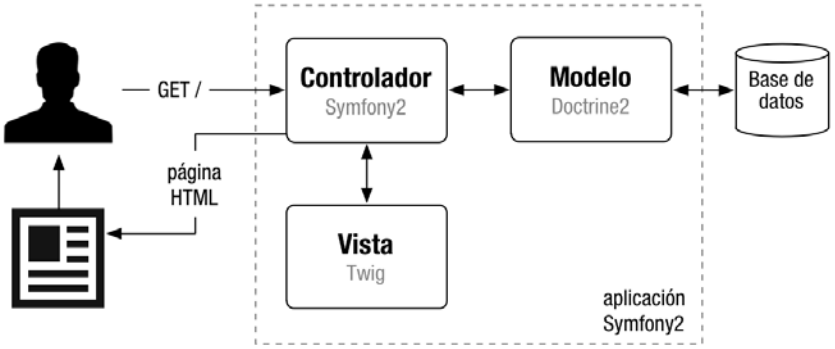


Figura 5: Arquitectura implementada por Symfony 2.

1.4.2. Vista física de la arquitectura del proyecto SIGEF II

El diagrama de despliegue es un diagrama que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. (45)

El diagrama de despliegue muestra la configuración de los nodos de procesamiento en tiempo de ejecución, los enlaces de comunicación entre ellos, y las instancias de los componentes y objetos que residen en ellos. El propósito es capturar la configuración de los elementos de procesamiento, y las conexiones entre estos elementos en el sistema. Los elementos que lo componen son:

- **Procesador:** nodo con elementos de procesamiento, al menos un procesador, memoria, y posiblemente otros dispositivos.
- **Dispositivos:** nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.
- **Conectores:** expresa el tipo de conector o protocolo utilizado entre el resto de los elementos del modelo.

A continuación se expone el diagrama de despliegue de la solución propuesta ya que la misma será usada en las dos instancias municipales y provinciales así como en la FGR:

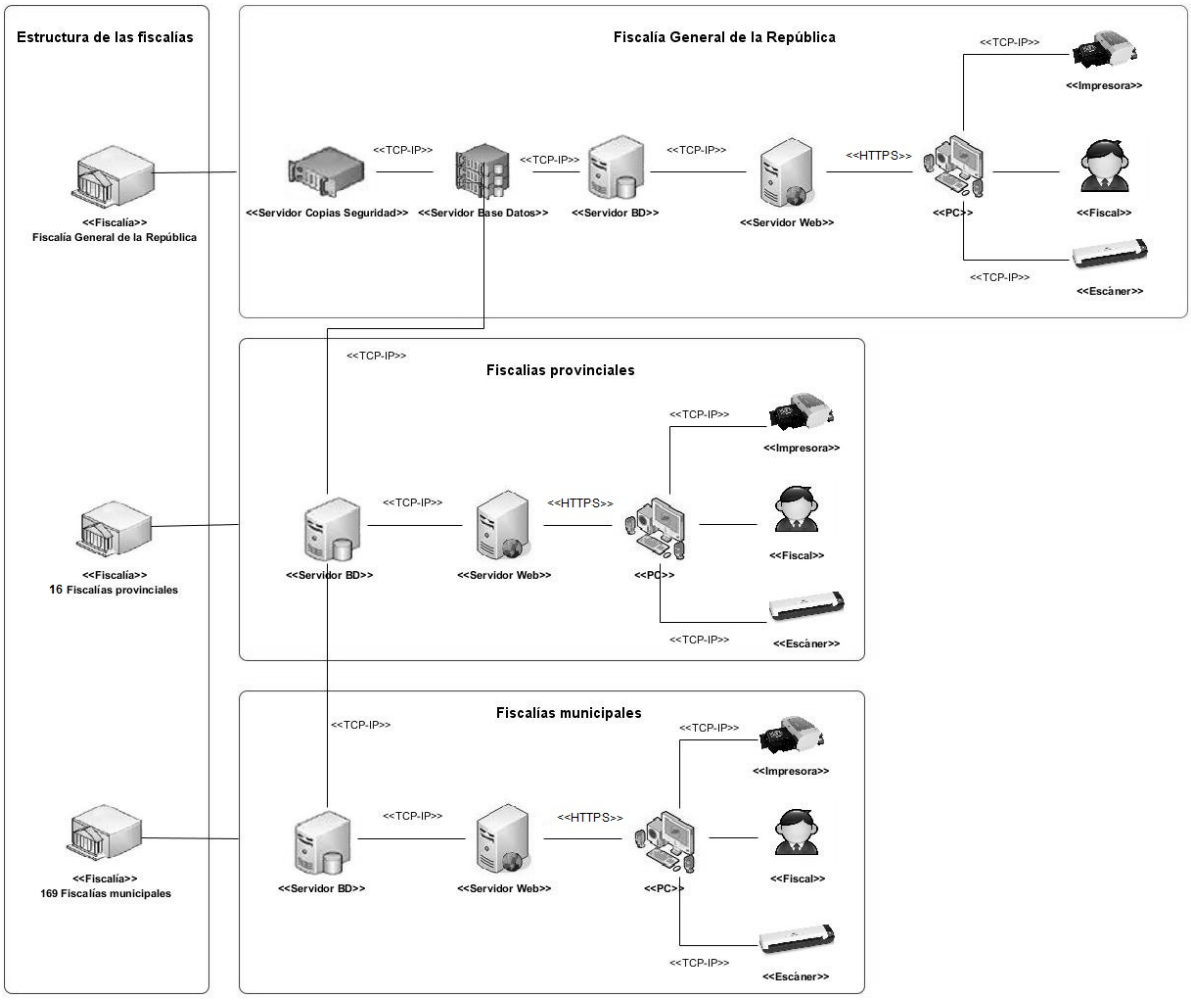


Figura 6: Modelo de despliegue del módulo Verificación Fiscal

La figura 5 muestra la distribución entre los nodos necesarios para el despliegue del sistema a nivel nacional. Se detalla la estructura de las diferentes instancias de las fiscalías cubanas, contando con ciento sesenta y nueve fiscalías municipales, dieciséis provinciales y una dirección general, la FGR. En cada instancia municipal y provincial se instalarán computadoras clientes, impresoras, escáneres, un servidor web y un servidor de base de datos; mientras en la FGR además de esos elementos se encontrará un servidor de base de datos encargado de la replicación entre servidores y un servidor de copias de seguridad para casos de fallo. El servidor de base de datos de las instancias municipales se conectará directamente con el de la fiscalía provincial correspondiente y los de las instancias provinciales a su vez, con el servidor de la FGR encargado de la replicación. La comunicación entre todos los nodos se realizará en su mayoría mediante el protocolo TCP/IP, a excepción de la comunicación entre las computadoras cliente y los servidores web que se comunicarán a través del protocolo HTTP.

1.4.3. Marco de trabajo

Symfony 2.1.7

Symfony 2 ha sido ideado para desarrollar aplicaciones comerciales y por eso es uno de los frameworks PHP con mejor rendimiento. Su arquitectura interna está completamente desacoplada, lo que permite reemplazar o eliminar fácilmente aquellas partes que no encajan en el proyecto. (16)

Entre sus características, Symfony tiene la de optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. (17)

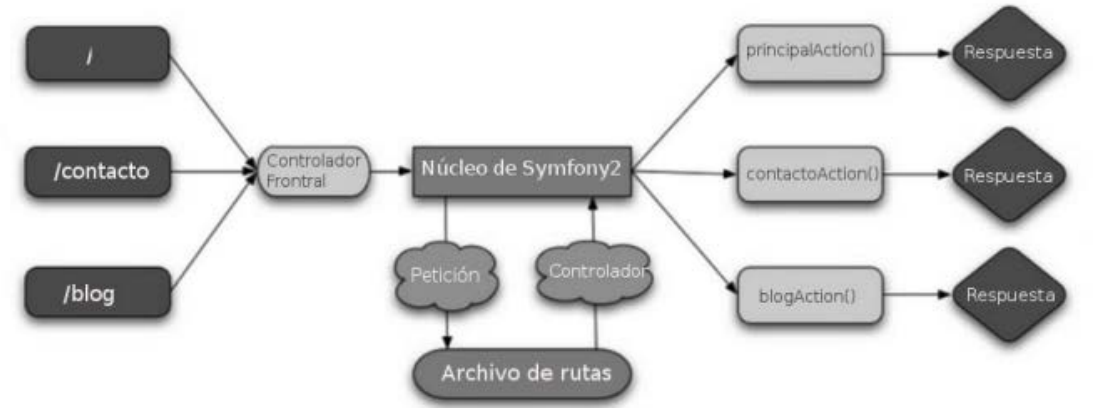


Figura 7: Arquitectura de symfony2.

Todas las peticiones de navegación del usuario son controladas mediante un controlador frontal, el cuál es el encargado de enviar la ruta que solicita el usuario hacia un controlador. Dicho controlador se encarga de manejar la lógica asociada a la petición del usuario y posteriormente envía la respuesta al usuario. (ver figura 7)

Symfony 2 es seleccionado porque es un marco de trabajo flexible ya que cada programador puede utilizarlo como desee, interoperable pues permite crear aplicaciones que se adapten exactamente a las necesidades de cada proyecto. Se escoge además este marco de trabajo por su abundante documentación, el potencial de trabajo que presenta para el desarrollo de aplicaciones web de forma rápida, robusta y segura. Se define utilizar

el marco de trabajo, ya que la primera fase del SIGEF fue desarrollada utilizando este marco de trabajo y para llevar una homogeneidad se decide utilizar la versión 2 del marco de trabajo, ya que el sistema es un sistema web.

Doctrine 2.3.0

El marco de trabajo Symfony 2 utiliza Doctrine como mapeador de objetos-relacional escrito en lenguaje PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un Sistema Gestor de Base de Datos (SGBD). Puede generar clases a partir de una base de datos existente y después se pueden especificar relaciones y añadir funcionalidades extras a las clases autogeneradas. (14)

Una característica importante de Doctrine es la posibilidad de escribir consultas de base de datos en un lenguaje de consulta estructurado (SQL por Structured Query Language) denominado lenguaje de consulta doctrine (DQL por Doctrine Query Language). (14)

Doctrine es seleccionado por la necesidad de utilizar un marco de trabajo que permita trabajar con los objetos de la base de datos. Doctrine viene integrado por defecto al marco de trabajo Symfony2, además de que permite la generación automática del modelo, logrando agilizar el trabajo. Para crear el modelo, doctrine genera una clase por tabla de base de datos e indica mediante PHP el tipo de datos que se almacenará.

1.4.4. Gestor de base de datos PostgreSQL 9.1.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, con su código fuente disponible libremente y distribuido bajo licencia BSD[1]. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multi-hilos para garantizar la estabilidad del sistema, un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. (18)

Su estabilidad, potencia, robustez, facilidad de administración han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. (19)

PostgreSQL es seleccionado porque es ideal para tecnologías web y fácil de administrar. Su código fuente es libre y de alta calidad, además de tener un rendimiento excelente.

[1] **Licencia BSD**: Berkeley Software Distribution (en español, distribución de software berkeley) es un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

Además permite gran capacidad de almacenamiento, requerida para la extensa información que gestiona SIGEF II.

1.4.5. Lenguajes utilizados

1.4.5.1. Lenguaje de Modelado (UML) 2.0

UML es un lenguaje de modelado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas, cuya finalidad es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. (8)

Se seleccionó UML como lenguaje para el modelado de diseños que traigan consigo la creación de una notación de diseño que sean válidas para los analistas, desarrolladores y clientes. Además es el lenguaje con el que trabaja el Visual Paradigm for UML.

1.4.5.2. Lenguajes de programación

Lenguajes de programación del lado del servidor

PHP 5.4.

PHP es un lenguaje interpretado de propósito general ampliamente usado, diseñado especialmente para desarrollo web y que puede ser incrustado dentro de código HTML. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. (20)

Es un lenguaje multiplataforma, completamente orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos. El código fuente escrito en PHP es invisible al navegador y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura. (20).

Para el desarrollo de la propuesta solución es seleccionado PHP 5.4 debido a que el marco de trabajo Symfony 2 necesita la versión 5.3 o una superior. Además es muy sencillo para el entendimiento de los desarrolladores.

1.4.5.3. Twig 1.12.1

Twig es un motor de plantillas para el lenguaje de programación PHP. Tiene un modo de recinto para evaluar código de la plantilla que no es de confianza. Es un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla y permite al desarrollador definir sus propias etiquetas y filtros personalizados. (21)

Este lenguaje permitió compilar las plantillas hasta código PHP optimizado, lo que proporcionó rapidez durante la implementación. Brindó seguridad, pues permite evaluar el código de plantilla que no es confiable. Es alimentado por flexibles analizadores: léxico y sintáctico, lo cual le posibilitó al desarrollador definir sus propias etiquetas y filtros personalizados.

Lenguajes de Programación del lado del cliente

1.4.5.4. JavaScript 1.6

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. 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. (22)

Es seleccionado este lenguaje pues existe la necesidad de validar los datos por el lado del cliente permitiendo crear efectos atractivos y dinámicos en las páginas web.

1.4.5.5. CSS 3

Es el acrónimo Cascade Style Sheet, traducido al español significa Hojas de Estilo en Cascada. Es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación. Es imprescindible para crear páginas web complejas. (23)

Permiten independizar el estilo del contenido del documento. Al independizar el estilo de la estructura, el acceso a ambas es mucho más rápido y flexible: se pueden definir varios estilos para un mismo documento, de forma que el usuario elija el que más le conviene. Mejora el rendimiento del ordenador, ya que la hoja de estilo se carga una única vez para todas las páginas html a las que se aplica. (24)

Se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página. Aporta muchas ventajas y facilidades a la hora de trabajar las plantillas que se van a mostrar en la aplicación, permitiendo ahorro de código y tiempo a los desarrolladores.

1.4.5.6. HTML 5

HTML 5 proporciona una plataforma con la que desarrollar aplicaciones web que permitan resolver las necesidades reales de los desarrolladores. Ayuda a los motores de búsqueda por medio de etiquetas informativas en la cabecera de cada página web, navegación,

artículo y pie de página; diciéndole a cada motor de búsqueda que tipo de contenido tiene y que enlace va a compartir. HTML 5 nos permite una mayor interacción entre nuestras páginas web facilitando la maquetación de las mismas. (25)

Es seleccionado ya que además de ser compatible con la mayoría de los navegadores, en términos de eficiencia y reducción de costos, el HTML5 brinda una serie de componentes que hacen que el trabajo sea más fácil para los desarrolladores pero también altamente efectivo para los usuarios.

1.4.6. Herramientas utilizadas

Se hizo necesario utilizar determinadas herramientas para facilitar el trabajo y la modelación de los procesos detección de violaciones y responsables. A continuación se detallan las características significativas que definieron su uso para el desarrollo del sistema.

Herramientas para el modelado. Visual Paradigm for UML 8.0.

Visual Paradigm para UML es una herramienta para desarrollo de aplicaciones utilizando modelado UML, ideal para ingenieros de software, analistas y arquitectos que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos. (26) Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Esta herramienta también proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML. (27) .Es muy completa y fácil de usar, con soporte multiplataforma y proporciona excelente facilidades de interoperabilidad con otras aplicaciones. Permite además invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. (28)

Se selecciona esta herramienta para agilizar el diseño ya que posee una interfaz fácil de usar. Además de ser un producto de alta calidad y prestaciones, la universidad cuenta con la posibilidad de tener su licencia de uso.

NetBeans 7.4.

NetBeans es un entorno de desarrollo integrado (IDE, por sus siglas en inglés), multiplataforma y de código abierto que también es compatible con otros lenguajes de programación.

No sólo permite diseñar y programar aplicaciones de escritorio, también se emplea para elaborar aplicaciones web que hagan uso de AJAX, CSS y JavaScript. Se integra al marco de trabajo Symfony2, a los demás lenguajes y herramientas propuestos que facilitan el desarrollo de aplicaciones de gran envergadura.

NetBeans fue seleccionado por la experiencia que tenía de su uso el equipo de desarrollo, lo que permitió agilizar el trabajo y lograr mayor productividad en el mismo. Permitió la rápida edición del código, sintáctica y semánticamente, la sencilla y eficiente gestión de la aplicación manteniendo un orden de las carpetas, archivos y líneas de código.

PgAdmin 3.

Esta herramienta posee una interfaz gráfica que soporta todas las características de PostgreSQL y facilita la administración, e incluso pueden realizarse cambios en el modelo de datos desde la herramienta de manera muy sencilla. PgAdmin 3 está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. (29)

PgAdmin 3 es seleccionado gracias a la fuerte integración que tiene con el Sistema Gestor de Base de Datos escogido, y por poseer características que permiten que el trabajo con los datos sea de manera fácil y sencilla.

Subversion 1.5.

Un sistema de control de versiones es un sistema para la gestión de archivos y directorios, que tiene como principal característica, mantener la historia de los cambios y modificaciones que se han realizado sobre ellos a lo largo del tiempo. (30)

Subversión fue la herramienta empleada para el control de versiones en el proyecto SIGEF II. Es uno de los sistemas de control de versiones que se ha popularizado dentro de la comunidad de desarrolladores de software libre. Garantizó atomicidad en las actualizaciones, permitió minimizar el riesgo de aparición de inconsistencias entre distintas partes del repositorio. Facilitó las acciones de añadir, eliminar y mover ficheros y directorios.

1.5. Diseño de software

Se define el diseño como un proceso o labor destinado a proyectar, coordinar, seleccionar y organizar un conjunto de elementos para producir y crear objetos visuales destinados a comunicar mensajes específicos a grupos determinados. (31)

1.5.1. Patrones

Un patrón se define como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución.

De forma general, un patrón sigue el siguiente esquema:

- ✓ **Contexto:** Es una situación de diseño en la que aparece un problema.
- ✓ **Problema:** Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- ✓ **Solución:** Es una configuración que equilibra estas fuerzas. (32)

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. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema, debido a que especifica a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema. (33)

Dentro de esta clasificación de patrones se encuentran los GRASP[2], estos representan los principios básicos de la asignación de responsabilidades a objetos y los GOF se utilizan en situaciones frecuentes, proponen soluciones a problemas concretos y se basan en la experiencia acumulada para resolver problemas reiterativos. En el capítulo 2 se muestra una explicación de los patrones de diseño que fueron empleados en la implementación de la solución.

1.6. Métricas de validación del diseño

Las métricas se centran en cuantificar tanto la complejidad, como la funcionalidad y eficiencia inmersa en el desarrollo de software. Inclina sus objetivos a mejorar la comprensión de la calidad del producto, a estimar la efectividad del proceso y mejorar la calidad del trabajo. (34)

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

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

[2] GRASP: Acrónimo para General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades)

- **Acoplamiento**. consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de reutilización.
- **Complejidad del mantenimiento**. consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas**. consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado. (35)

A continuación se explican dos de las métricas de diseño que permiten evaluar estos atributos.

Tamaño operacional de clase (TOC)

El tamaño de la clase se determina con la suma del número de atributos (tanto atributos heredados como atributos privados de la instancia) que están encapsulados en la clase y el número total de operaciones (tanto operaciones heredadas como operaciones privadas de la instancia) que están encapsuladas dentro de la clase (36). Los umbrales para medir el TOC son:

Tabla 2: clasificación del TOC de las clases

Clasificación	Valores de los umbrales
Pequeño	<=20
Medio	>20 <=30
Grande	>30

Si existen valores grandes de TOC demostrarán que una clase puede tener demasiada responsabilidad, lo cual reduciría la reutilización de la clase y hará complicada la implementación.

Tabla 3: Modo en que afecta el TOC a los atributos de calidad.

Atributo que afecta	Modo en que lo afecta
Responsabilidad	Un aumento del TOC indica que la clase puede tener demasiada responsabilidad.
Complejidad de	Un aumento del TOC implica un aumento de la

Implementación	complejidad de implementación de la clase.
Reutilización	Un aumento del TOC reduce la reutilización de la clase.

Para determinar el valor de los atributos de calidad, se calcula el promedio de la columna cantidad de procedimientos y se emplea en la tabla 4 en la columna criterio.

Tabla 4: Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica

Atributo	Categoría	Criterio
Responsabilidad	Baja	CP <= Promedio
	Media	Promedio <= CP <= 2* Promedio
	Alta	CP > 2* Promedio
Complejidad de Implementación	Baja	CP <= Promedio
	Media	Promedio <= CP <= 2* Promedio
	Alta	CP > 2* Promedio
Reutilización	Baja	CP > 2* Promedio
	Media	Promedio <= CP <= 2* Promedio
	Alta	CP <= Promedio

Relaciones entre clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otras. (10) A continuación se muestran las tablas encaminadas a un mejor entendimiento de la utilización de esta métrica.

Tabla 5: Modo en que afectan las RC a los atributos de calidad

Atributo que afecta	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.
Cantidad de pruebas	Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Para determinar el valor de los atributos de calidad, se calcula el promedio de asociaciones de uso y se emplea en la tabla 6 en la columna criterio.

Tabla 6: Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	Cantidad de relaciones de uso > 2
Complejidad de mantenimiento	Baja	Cantidad de relaciones de uso <= Promedio
	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio
Cantidad de pruebas	Baja	Cantidad de relaciones de uso <= Promedio
	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio

1.7. Pruebas para la validación del sistema

El único instrumento adecuado para determinar la calidad de un producto software es el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. En las pruebas se usan casos de prueba, especificados de forma estructurada mediante técnicas de prueba. (37) Son actividades que tienen como objetivo identificar posibles fallos de implementación, calidad o usabilidad en un sistema. Los resultados se observan, registran y se realiza una evaluación de los aspectos detectados.

1.7.1. Pruebas de unidad

Son procedimientos de pruebas locales a un módulo del sistema. Dichas pruebas cubren las funcionalidades propias del módulo prestando poca o ninguna atención a la integración con otros módulos. (38) A continuación se detallan los enfoques de estas pruebas:

Pruebas de caja blanca.

Las pruebas de caja blanca son pruebas estructurales que conociendo el código y siguiendo su estructura lógica, se pueden diseñar pruebas destinadas a comprobar que el código hace correctamente lo que el diseño indica y otras que demuestren que no se

comporta adecuadamente ante determinadas situaciones. Ejemplos típicos de ello son las pruebas unitarias. Se centran en lo que hay codificado o diseñado a bajo nivel por lo que no es necesario conocer la especificación de requisitos, que por otra parte será difícil de relacionar con partes diseñadas a muy bajo nivel. (39)

Técnica del camino básico: permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

A continuación se describen los pasos específicos para realizar esta técnica en relación a los mencionados anteriormente para un mejor entendimiento del algoritmo:

1. **Notación del grafo de flujo:** usando el código como base, se realiza la representación del grafo de flujo (grafo del programa) mediante una sencilla notación. Cada construcción estructurada tiene su correspondiente símbolo.
   - ❖ Nodo: cada círculo, denominado nodo, representa una o más sentencias procedimentales.
   - ❖ Arista: las flechas del grafo de flujo, denominadas aristas, representan el flujo de control.
   - ❖ Región: las áreas delimitadas por aristas y nodos se denominan regiones.
2. **Complejidad ciclomática:** es una métrica que proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado define el número de caminos independientes del conjunto básico de un programa y propone que se realice al menos un caso de prueba por cada camino independiente, de manera que se asegure la ejecución de cada sentencia como mínimo una vez.

   Un camino independiente es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición. En términos del grafo de flujo, un camino independiente está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino. (10)

La complejidad ciclomática V (G), se calcula de tres formas:

- Número de regiones del grafo de flujo.
- 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) = P+ 1, donde P es el número de nodos predicado contenidos en el grafo de flujo. Un nodo predicado es aquel que contiene una condición, y está caracterizado porque dos o más aristas emergen en él.

3. Determinar un conjunto básico de caminos linealmente independientes: el valor de V (G) coincide con el número de caminos linealmente independientes de la estructura de control del programa.

4. Obtención de casos de prueba: se realizan los casos de pruebas que forzarán la ejecución de cada camino del conjunto básico.

Para la realización de las pruebas de caja blanca se utilizó la herramienta PHPUnit.

Herramienta PHPUnit 3.4

PHPUnit es un framework PHP para realizar pruebas unitarias. Permite realizar las pruebas pertinentes al código, verificando que el funcionamiento de las aplicaciones PHP es el deseado y en su caso encontrando errores que una vez solucionados mejorarán la calidad de nuestros desarrollos web. La actual versión PHPUnit 3.4 requiere PHP 5.1.4 o superior.

Algunos de los beneficios de PHPUnit son:

- Reducción del esfuerzo necesario para poner a prueba con frecuencia un mismo código
- Resultados más robustos en sus aplicaciones
- Mayor confianza en el código

1.7.2. Pruebas de sistema

Pruebas de caja negra

Son pruebas funcionales. Se parte de los requisitos funcionales, a muy alto nivel, para diseñar pruebas que se aplican sobre el sistema sin necesidad de conocer como está construido por dentro. Las pruebas se aplican sobre el sistema empleando un determinado conjunto de datos de entrada y observando las salidas que se producen para determinar si la función se está desempeñando correctamente por el sistema bajo prueba. Las herramientas básicas son observar la funcionalidad y contrastar con la especificación. (39)

Técnica de la Partición de Equivalencia: es una de las más efectivas pues permite examinar los valores válidos y no válidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así en número de clases de prueba que hay que desarrollar. (40)

1.8. Conclusiones parciales

- Al realizar el análisis de la metodología de desarrollo utilizada en el proyecto se pudo conocer los artefactos necesarios para la ejecución de los flujos de trabajo Diseño, Implementación y Pruebas, así como los artefactos de salida.
- El estudio de la arquitectura del proyecto permitió conocer los lenguajes y herramientas a utilizar para el diseño y la implementación de los procesos Detección de violaciones y responsables de la legalidad.
- Se estudiaron las métricas RC y TOC que permitirán validar el diseño, así como las técnicas de pruebas a aplicar para la validación de la aplicación.

CAPITULO 2: DISEÑO E IMPLEMENTACIÓN

2.1. Introducción

En el presente capítulo se reflejan los requisitos no funcionales definidos para el sistema y se hace referencia a los requisitos funcionales pertenecientes a los procesos violaciones y responsables. Además se presenta el diseño realizado asociado a dichos requisitos, el cual abarca los diagramas de clases necesarios para los procesos asociados a las violaciones y responsables del módulo VF. Posteriormente se muestra como fue concebido el desarrollo de todo el trabajo correspondiente con la implementación del sistema en términos de componentes.

2.2. Elementos de entrada al diseño

Durante el desarrollo de un software el diseño se realiza con el fin de traducir los requisitos a una estructura que describe cómo implementar el sistema, siendo lo suficientemente específica para que no existan ambigüedades y ofreciendo la posibilidad de descomponer los trabajos de implementación en componentes manejables, visualizándolos mediante una notación común. Para lograr una mejor comprensión del diseño e implementación del sistema a desarrollar se muestran a continuación los principales artefactos obtenidos en el análisis por parte del equipo de analistas del proyecto SIGEF II.

2.2.1. Requisitos funcionales

Los requisitos funcionales son una descripción de las necesidades o cualidades de un producto. La meta primaria de la fase de requerimientos es identificar y documentar lo que en realidad se necesita, en una forma que claramente se le comunique al cliente y a los miembros del equipo de desarrollo. (41)

Los requisitos funcionales son los que definen el comportamiento interno del sistema. Abarcan las funcionalidades específicas que el software debe ser capaz de realizar. (42)

A continuación las agrupaciones funcionales que están integradas por los requisitos funcionales que serán implementados, correspondientes a los procesos Violaciones y Responsables del módulo VF. El término agrupación funcional es tomado por el equipo de análisis con el fin de agrupar por concepto los requisitos asociados a una funcionalidad determinada.

Tabla 7 : Agrupaciones Funcionales

No	Agrupaciones Funcionales	Complejidad	Prioridad	Requisitos
1	Gestionar presuntas violaciones	Media	Media	4
2	Gestionar violaciones de la legalidad	Alta	Alta	16

3	Gestionar denuncia	Baja	Media	3
4	Gestionar informe de denuncia de hechos delictivos a la PNR	Media	Media	4
5	Gestionar informe de denuncia de hechos delictivos al órgano de Instrucción	Media	Media	4
6	Gestionar informe de comunicación de indicios de enriquecimiento indebido	Media	Baja	5
8	Gestionar responsable de la violación	Alta	Alta	17
9	Gestionar responsable de la acción	Baja	Baja	1

Tabla 8: Requisitos funcionales asociados a cada agrupación funcional

Gestionar Presuntas Violaciones	
RF_VF_VF_1	Listar presuntas violaciones de la legalidad
RF_VF_VF_2	Adicionar presunta violación de la legalidad
RF_VF_VF_3	Actualizar presunta violación de la legalidad
RF_VF_VF_4	Visualizar detalles de la presunta violación de la legalidad
Gestionar Violaciones de la legalidad	
RF_VF_VF_5	Listar violaciones de la legalidad
RF_VF_VF_6	Adicionar violación de la legalidad
RF_VF_VF_7	Actualizar violación de la legalidad
RF_VF_VF_8	Eliminar violación de la legalidad
RF_VF_VF_9	Detalles de la violación de la legalidad
RF_VF_VF_10	Listar normas legales infringidas
RF_VF_VF_11	Adicionar normas legales infringidas
RF_VF_VF_12	Actualizar normas legales infringidas
RF_VF_VF_13	Eliminar normas legales infringidas
RF_VF_VF_14	Visualizar detalles de las normas legales infringidas
RF_VF_VF_15	Listar afectaciones económicas
RF_VF_VF_16	Adicionar afectación económica
RF_VF_VF_17	Actualizar afectación económica
RF_VF_VF_18	Eliminar afectación económica
RF_VF_VF_19	Listar elementos de prueba
RF_VF_VF_20	Eliminar elementos de prueba

Gestionar Denuncia	
RF_VF_VF_21	Listar denuncias
RF_VF_VF_22	Adicionar denuncia
RF_VF_VF_23	Actualizar denuncia
Gestionar informe de denuncia de hechos delictivos a la PNR	
RF_VF_VF_24	Listar informes de denuncia de hechos delictivos a la PNR
RF_VF_VF_25	Adicionar informe de denuncia de hechos delictivos a la PNR
RF_VF_VF_26	Actualizar informe de denuncia de hechos delictivos a la PNR
RF_VF_VF_27	Vista previa del informe de denuncia de hechos delictivos a la PNR
RF_VF_VF_28	Listar Informes de denuncia de hechos delictivos al órgano de instrucción
RF_VF_VF_29	Adicionar Informe de denuncia de hechos delictivos al órgano de instrucción
RF_VF_VF_30	Actualizar Informe de denuncia de hechos delictivos al órgano de instrucción
RF_VF_VF_31	Vista previa del informe de denuncia de hechos delictivos al órgano de instrucción
Gestionar Informe de Comunicación de indicios de enriquecimiento indebido	
RF_VF_VF_32	Listar informes de comunicación de indicios de enriquecimiento indebido
RF_VF_VF_33	Adicionar Informe de comunicación de indicios de enriquecimiento indebido
RF_VF_VF_34	Actualizar Informe de comunicación de indicios de enriquecimiento indebido
RF_VF_VF_35	Vista previa del Informe de comunicación de enriquecimiento indebido
RF_VF_VF_36	Adicionar constancia de aprobación del informe de comunicación de enriquecimiento indebido.
Gestionar Responsable de la violación	
RF_VF_VF_37	Listar responsables de la violación

RF_VF_VF_38	Actualizar responsables de la violación
RF_VF_VF_39	Detalles de los responsables de la violación
RF_VF_VF_40	Listar responsabilidades
RF_VF_VF_41	Adicionar responsabilidad
RF_VF_VF_42	Actualizar responsabilidad
RF_VF_VF_43	Eliminar responsabilidad
RF_VF_VF_44	Visualizar detalles de la responsabilidad administrativa
RF_VF_VF_45	Visualizar detalles de la responsabilidad material
RF_VF_VF_46	Visualizar detalles de la responsabilidad penal.
RF_VF_VF_47	Listar afectación material
RF_VF_VF_48	Actualizar afectación material
RF_VF_VF_49	Eliminar afectación material
RF_VF_VF_50	Listar afectación penal
RF_VF_VF_51	Actualizar afectación penal
RF_VF_VF_52	Eliminar afectación penal
RF_VF_VF_53	Eliminar norma legal infringida por el responsable
Gestionar responsable de la acción	
RF_VF_VF_54	Listar responsables de la acción

2.2.2. Especificación del requisito Adicionar presunta violación de la legalidad

Seguidamente se muestra la especificación de uno de los requisitos funcionales de prioridad alta, que guiará las labores de implementación del mismo. Las restantes especificaciones de los requisitos funcionales pueden ser consultadas en el documento CEGEL_SIGEFII_0113_ERS_VF_VF.

Tabla 9: Especificación del requisito Adicionar presunta violación de la legalidad.

Nº	***Nombre***	***Descripción***	***Complejidad***	***Prioridad para el cliente***
RF_VF_VF_138	Adicionar presunta violación de la legalidad	Permite adicionar una presunta violación de la legalidad.	Baja	Alta
	Prototipo			

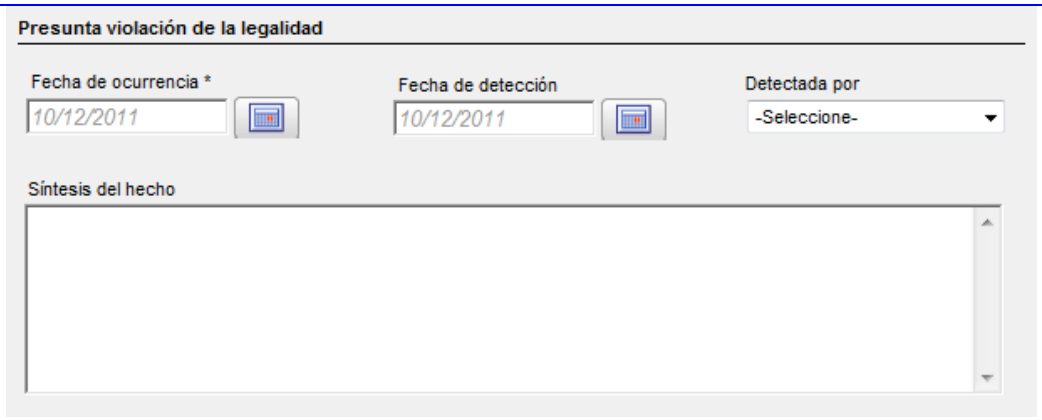


Figura 138.1 Adicionar presunta violación de la legalidad.

Campos	***Tipos de Datos***	***Reglas o Restricciones***
Fecha de ocurrencia	Cadena	Opcional
Fecha de detección	Cadena	Obligatorio. Este campo debe tener un valor menor o igual a la fecha actual.
Detectada por	Cadena	Obligatorio. Este campo puede tomar los siguientes valores: • *Fiscal actuante* • *Especialista adjunto* • *Especialista de la fiscalía*
Síntesis del hecho	Cadena	Obligatorio.
Observaciones	Se insertan además los participantes de la presunta violación, ver requisito *RF_VF_VF_140*.	

Se describe a continuación la funcionalidad Adicionar presunta violación de la legalidad, a la cual se le dará seguimiento a lo largo del trabajo de diploma.

Un hecho o también llamado presunta violación es uno de los procesos que se realizan en las verificaciones fiscales y que dan paso al proceso violaciones. En el momento que se detecta alguna irregularidad dentro de una entidad objeto de una verificación se está en presencia de un hecho o una presunta violación porque puede constituir una violación o no. Se convierte en una violación cuando se infringe en una norma legal registrada en la ley. La funcionalidad Adicionar presunta violación de la legalidad consiste en la creación de los

hechos. Para adicionar un hecho al sistema, el usuario primeramente debe partir de una pantalla inicial que contiene las acciones que pueden ser investigaciones o verificaciones fiscales. Al seleccionar una acción determinada, aparecerá una interfaz con todos los hechos asociados a dicha acción creados hasta el momento. Al elegir la opción Adicionar, aparecerá una interfaz que contiene los datos fundamentales del hecho que se está creando, entre los cuales se encuentran la fecha de detección, la fecha en que ocurrió, la persona que lo detectó y la síntesis del mismo. Luego, el usuario procede a buscar a las personas y/o entidades que están asociadas al proceso, las cuales ocupan el rol de participante

2.2.3. Requisitos no funcionales

A continuación se muestran algunos de los requisitos no funcionales que se evidencian en el sistema, con una breve descripción:

Usabilidad:

RnF 1. La aplicación informática a desarrollar será una aplicación web.

RnF 2. La aplicación informática a desarrollar será un software de gestión.

RnF 3. Entorno de producción de la aplicación informática a desarrollar.

Confiabilidad:

RnF 4. Fiabilidad del sistema. Existirán servidores locales con capacidad necesaria para el procesamiento de las solicitudes del conjunto de aplicaciones de las diferentes oficinas.

RnF 5. Disponibilidad del sistema. El sistema estará disponible durante el horario laboral, efectuándose en períodos de tiempos definidos el proceso de actualización de la información del servidor local de base de datos con el Centro de Datos.

RnF 6. Exactitud de las respuestas. Las salidas que ofrece el sistema como resultado del manejo y procesamiento de la información estarán adecuadamente validadas durante los hitos de prueba.

Eficiencia:

RnF 7. Tiempo de respuesta. La mayoría de los procesos que se implementan con transacciones donde se modifica la base de datos deben tener tiempos de respuesta no mayores a los 3 segundos

Soporte:

RnF 8. Reparabilidad, mantenibilidad, escalabilidad. Para garantizar estas características se desarrollará el sistema orientado a componentes.

Interfaz:

RnF 9. Interfaz de usuario. El sistema tiene que ofrecer una interfaz amigable y debe ostentar un diseño sencillo, con pocas entradas, permitiendo un balance adecuado entre funcionalidad y simplicidad.

RnF 10. Interfaz de hardware. Las estaciones de trabajo pueden acceder a impresoras dependiendo de la funcionalidad establecida.

Seguridad:

RnF 11. Autenticación. Se establecerán mecanismos de autenticación personalizada para todos los usuarios que harán uso de la aplicación mediante dirección o rango de direcciones IP autorizadas a hacer uso del sistema.

RnF 12. Autorización. Existirán diferentes tipos de usuarios según las acciones que puedan realizar. Los permisos serán limitados basados en los roles definidos y el usuario no podrá gestionarlos.

RnF 13. Confidencialidad e integridad de los datos. Garantizar el uso de servicios de red, puertos y protocolos compatibilizados con las políticas de seguridad trazados por la entidad que soportará el despliegue de la aplicación.

Se identificaron un total de 19 requisitos no funcionales. Para acceder a la especificación de los mismos ver el documento 0113_Especificación de Requisitos de Software SIGEF II.

2.3. Diseño del sistema

A partir de los requisitos funcionales el diseño pretende formular los modelos que van a representar el dominio de la solución propuesta. De esta manera será más fácil descomponer el trabajo de la implementación que se pueda realizar.

2.3.1. Diagramas de clases

El modelo de diseño es una abstracción del modelo de implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño. Es usado como entrada esencial en las actividades relacionadas a implementación. (43) El documento 0121_Modelo de Diseño v2.0 contiene los diagramas y la descripción de las clases del diseño del módulo VF de SIGEF II. Seguidamente se mostrarán los diagramas del diseño generados para la funcionalidad Adicionar presunta violación de la legalidad.

Diagrama de clases persistentes.

El diagrama de clases persistentes brinda una vista de la solución final y resulta mucho más fácil efectuar cambios en el mismo, una vez detectados los problemas, que tener que efectuar cambios a la solución ya implementada. Permite reflejar las clases con las cuales

se va a trabajar, ya que recogen toda la información necesaria, y por lo tanto van a persistir en la base de datos.

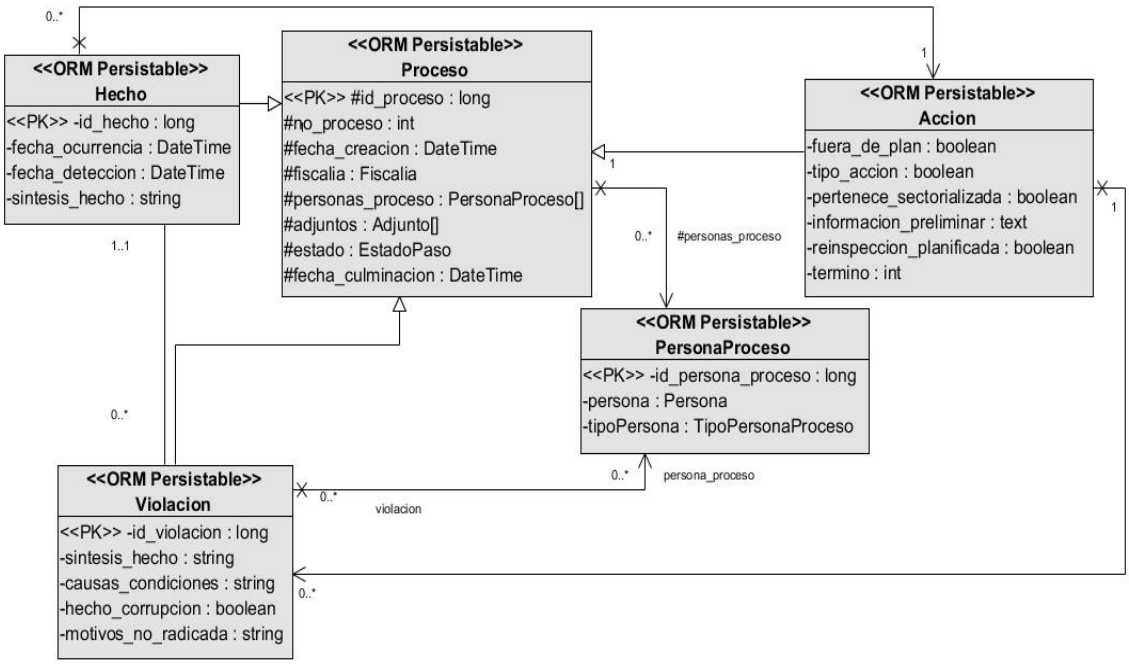


Figura 8: Diagrama de clases persistentes para la funcionalidad Adicionar presunta violación de la legalidad.

La figura 8 muestra el diagrama de clases persistentes correspondiente a la funcionalidad Adicionar presunta violación de la legalidad. En el proyecto SIGEF II existen clases comunes de las cuales los módulos se apropian de las características para llevar a cabo cada proceso, por ejemplo: las clases Proceso y PersonaProceso. La clase principal es Hecho, clase que se relaciona con una Acción porque parte fundamentalmente de la misma. Además se relaciona con una Violación porque puede ser una o no.

Diagrama de clases controladoras y gestoras.

Las clases controladoras (Controller) manejan el trabajo de las clases, encapsulan el comportamiento de una funcionalidad, manejan el flujo entre la vista y el modelo, además de realizar las operaciones más complejas. Las clases gestoras (Gtr) son las intermediarias entre la clase controladora y las clases repositorio, se ocupan de la lógica del negocio, implementando funcionalidades que garantizan el cumplimiento de los requisitos identificados. Estos tipos de clases se incluyen en el mismo diagrama de acuerdo a la estrecha relación que existe entre ellas a la hora de satisfacer cualquier funcionalidad. Se realizó este diagrama para tener definidos los métodos necesarios para la realización de la aplicación sin tener que recurrir a los métodos definidos en el módulo común.

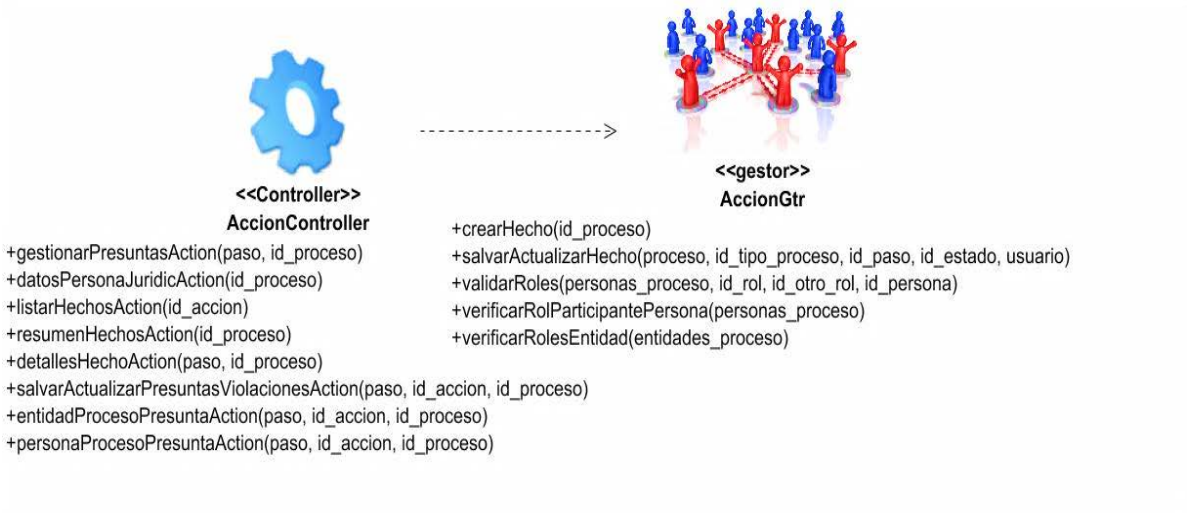


Figura 9: Diagrama de clases controladoras gestoras de la funcionalidad "Adicionar presunta violación de la legalidad."

La figura 9 muestra las clases *AccionController* y *AccionGtr*. La clase *AccionController* se encarga de dar respuesta a las peticiones de los usuarios, solicitando los datos a la clase *AccionGtr*, que se ocupa de gestionar la información relativa al negocio, como por ejemplo, la creación de instancias de las entidades además de la validación de las personas asociadas al proceso.

Diagrama de clases repositorio.

El diagrama de repositorios se emplea para representar las clases encargadas de realizar las consultas a la base de datos, elemento de gran importancia para el marco de trabajo seleccionado por el proyecto.

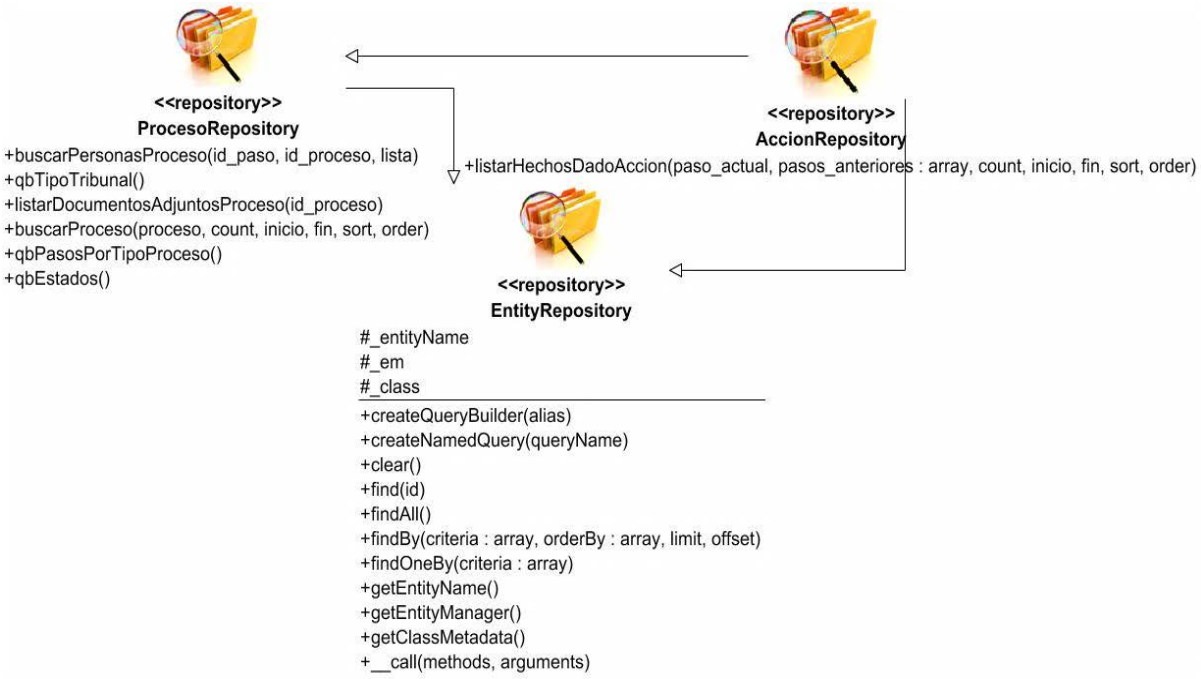


Figura 10: Diagrama de clases repositorios del diseño de la funcionalidad "Adicionar presunta violación de la legalidad"

En la figura 10 es mostrada la relación de herencia existente entre la clase del módulo *AccionRepository* y las clases *ProcesoRepository* y *EntityRepository*. También se evidencia la operación a realizar por la clase del módulo (*AccionRepository*), la cual permite obtener todos los hechos de una acción recepcionados hasta el momento. Se pueden realizar las consultas *find (), findAll (), findBy () y findOneBy ()*. El inconveniente de estos métodos es que no permiten reutilizar búsquedas complejas en diferentes partes de la aplicación. Para solucionarlo, es necesario crear un repositorio propio para añadir nuevos métodos de búsqueda.

Diagrama de clases de la vista.

Un diagrama de vistas representa la capa de presentación del sistema. En él se muestran las relaciones existentes entre las clases de la vista, posibilitando un mejor entendimiento de las interfaces a implementar. En el siguiente diagrama de vista se evidencia la secuencia de interfaces de usuario y los formularios que se visualizan en una determinada acción.

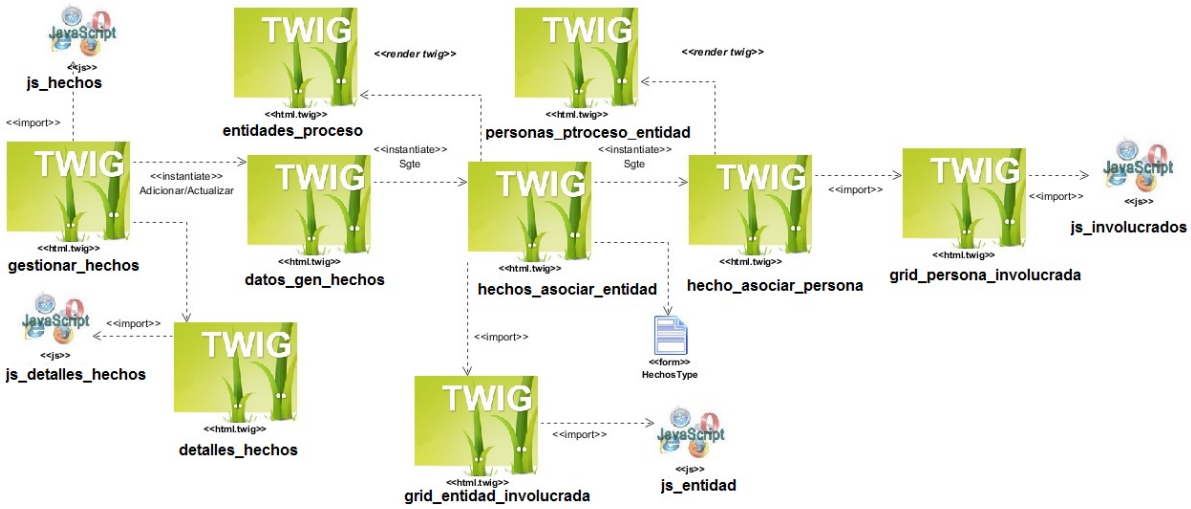


Figura 11: Diagrama de vista para la funcionalidad "Adicionar presunta violación de la legalidad"

En la figura 11 se muestra cómo se relacionan las clases Twig con los distintos formularios y las clases JavaScript en la funcionalidad Adicionar presunta violación de la legalidad. Este diagrama tiene 9 clases Twig, de estas, dos son parte del subsistema base. También componen este diagrama un formulario y cuatro archivos JavaScript.

2.3.2. Patrones de diseño empleados

Para diseñar el módulo Verificación Fiscal se emplearon un conjunto de patrones de diseño que constituyen aspectos claves y que responden a soluciones simples ya que se reducen los esfuerzos de desarrollo garantizando eficiencia y consistencia en el diseño.

Los patrones GRASP son empleados en su totalidad, mientras que Symfony, como marco de trabajo empleado por el proyecto, utiliza varios patrones pertenecientes a los GOF, de forma que se promuevan las buenas prácticas de diseño y programación, logrando un mejor funcionamiento del sistema.

2.3.2.1. Patrones de diseño GRASP

- ✓ **Alta cohesión:** Su uso esta evidenciado en la implementación de las clases que forman parte de las capas controladora, negocio y modelo; las cuales están formadas por diferentes funcionalidades que se encuentran estrechamente relacionadas. (ver Figura 9) Diagrama de clases controladoras y gestoras para la funcionalidad "Adicionar presunta violación de la legalidad".
- ✓ **Creador:** El patrón creador ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos. En la solución propuesta fue utilizado el patrón en las clases gestora *AccionGtr* para crear las entidades y la clase controladora *AccionController* para crear los formularios.

```
public function salvarActualizarPresuntasViolacionesAction($paso,$id_accion,$id_proceso){
    $peticion = $this->getRequest();
    $proceso = $this->getAccionGtr()->crearHecho($id_accion,$id_proceso);
    $frm = $this->createForm(new PresuntasViolacionesType(),$proceso);
```

Figura 12 : Ejemplo del patrón creador de la clase *AccionController*.

- ✓ **Experto**: Se evidencia con el uso de clases que cuentan con la información necesaria para cumplir responsabilidades propias con relación al tipo de información que proporcionan. El uso de este patrón se evidencia en las clases repositorio, que son expertas en realizar operaciones de acceso a datos solo con la entidad que representa.(ver figura 13)

Figura 13: Clase encargada de listar las presuntas violaciones de un acción.

- ✓ **Bajo acoplamiento:** Es el encargado de asignar una responsabilidad para conservar bajo acoplamiento lo cual significa mantener las clases con la menor cantidad de relaciones posibles y de esta manera los componentes no se afectan por cambios de otros componentes, son fáciles de entender por separado y fáciles de reutilizar. Este patrón está evidenciado en el marco de trabajo ya que dentro de la capa modelo las clases de abstracción de datos son las más reutilizables y no tienen asociaciones con las clases de la capa vista ni con el controlador.
- ✓ **Controlador:** El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. (10) Este patrón se puede observar por lo general en la clase controladora *AccionController* y en el controlador frontal de Symfony (app.php): siendo este el único punto de entrada a la aplicación en un entorno determinado.

```
if ($peticion->getMethod() == 'POST') {
    $frm->bind($peticion);
    $proceso = $this->getAccionGtr()->salvarActualizarProceso($proceso, ConfigUtilVF:: Hecho_VF, $paso,
    ConfigUtilVF::estado_edicion, $usuario);
    return $this->render('VFBundle:Violacion:adicionar_presunta_violacion.html.twig', array(
        'frm' => $frm->createView(),
        'paso' => $paso,
        'id_accion' => $id_accion,
        'id_proceso' => $id_proceso));
```

Figura 14: Ejemplo de uso del patrón controlador en la clase *AccionController*.

2.3.2.2. Patrones de diseño GoF

Los patrones GoF (Gang of Four, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) se clasifican en 3 categorías basadas en su propósito:

Patrones de comportamiento: Más que describir objetos o clases, describen la comunicación entre ellos.

Patrones estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.

Patrones creacionales: Se encargan de la inicialización y configuración de objetos. (44)

El marco de trabajo Symfony2 con el cual se implementará el sistema a desarrollar ya trae consigo implementados muchos de estos patrones en su sistema interno, dándole solución a muchos problemas que se presentan continuamente e implementando así numerosas funcionalidades para facilitar el trabajo de los desarrolladores.

Patrones de comportamiento

Observador: Este patrón es uno de los patrones que trae por defecto el marco de trabajo Symfony 2. Es un patrón que define una dependencia entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes. El despachador de eventos de Symfony2 (ver figura 15) implementa el patrón observador en una manera sencilla y efectiva. Una vez creado un objeto Respuesta, puede ser útil permitir que otros elementos en el sistema lo modifiquen (por ejemplo, añadan algunas cabeceras de caché) antes de utilizarlo realmente. Para hacer esto posible, el núcleo de Symfony2 lanza un evento "*kernel.response*".

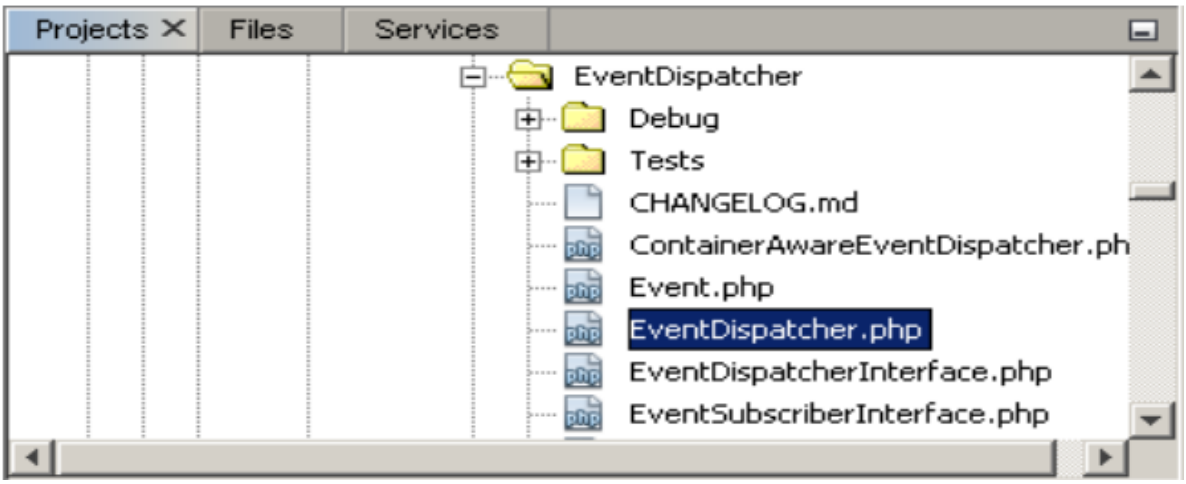


Figura 15: Paquete despachador de eventos de Symfony 2.

Patrones Creacionales

Fabrica Abstracta: Su propósito es definir una interfaz para la creación de familias de objetos relacionados o dependientes sin tener que especificar la clase concreta. La fábrica abstracta se utiliza para la creación de los objetos entidades y en la creación de los

formularios del sistema. Se potencia el encapsulamiento, puesto que se aísla a los clientes de las implementaciones y se obtiene un incremento de la flexibilidad del diseño. Ejemplo de su uso se aprecia en la creación de los formularios en la clase *AccionController*.

```
$frm = $this->createForm(new PresuntasViolacionesType(),$proceso);
```

Figura 16: Creación de formularios en la clase *AccionController*

Patrones Estructurales

Decorador: Es aplicado a la generación de vistas. La solución que ofrece dicho patrón es la de añadir funcionalidad adicional a las plantillas. Esto permite fragmentar la vista en distintas plantillas organizadas por criterios funcionales, y combinarlas para producir la vista completa. Este patrón se observa en el archivo denominado *layoutVF.html.twig* que contiene el layout del módulo Verificación Fiscal. El mismo almacena el código HTML que es común para todas las páginas, por lo que cada página que se cree heredará de esta, para no tener que repetir el código, y de esta forma el layout es quien decora la plantilla.

Figura 17: Uso del patrón decorador en la herencia de plantillas Twig.

Fachada: Se pone de manifiesto en la clase *AccionGtr*, la misma se encarga de definir un método capaz de salvar o actualizar determinado objeto, dicho método funciona como intermediario entre esta clase y los métodos propios de Symfony, que son los que realmente obtienen todos los datos correspondientes para persistir el objeto en la base de datos. Fundamentalmente se ve evidenciado en esta clase gestora ya que es intermediaria entre la clase controladora y la clase repositorio, se ocupa de la lógica del negocio, implementando funcionalidades que garantizan el cumplimiento de los requisitos identificados.

2.3.3. Diagramas de Secuencia

Los diagramas de secuencias son utilizados para modelar los aspectos dinámicos de los sistemas, muestran la realización de un flujo o escenario concreto de un requisito en términos de interacción entre objetos del diseño. Se les atribuye una gran importancia debido a la comprensión del sistema a la que puede llegarse a través de ellos. (10) A

continuación se muestra el diagrama de secuencia relacionado con la funcionalidad Adicionar Presuntas Violaciones el cual representa el flujo para registrar los hechos o presuntas violaciones.

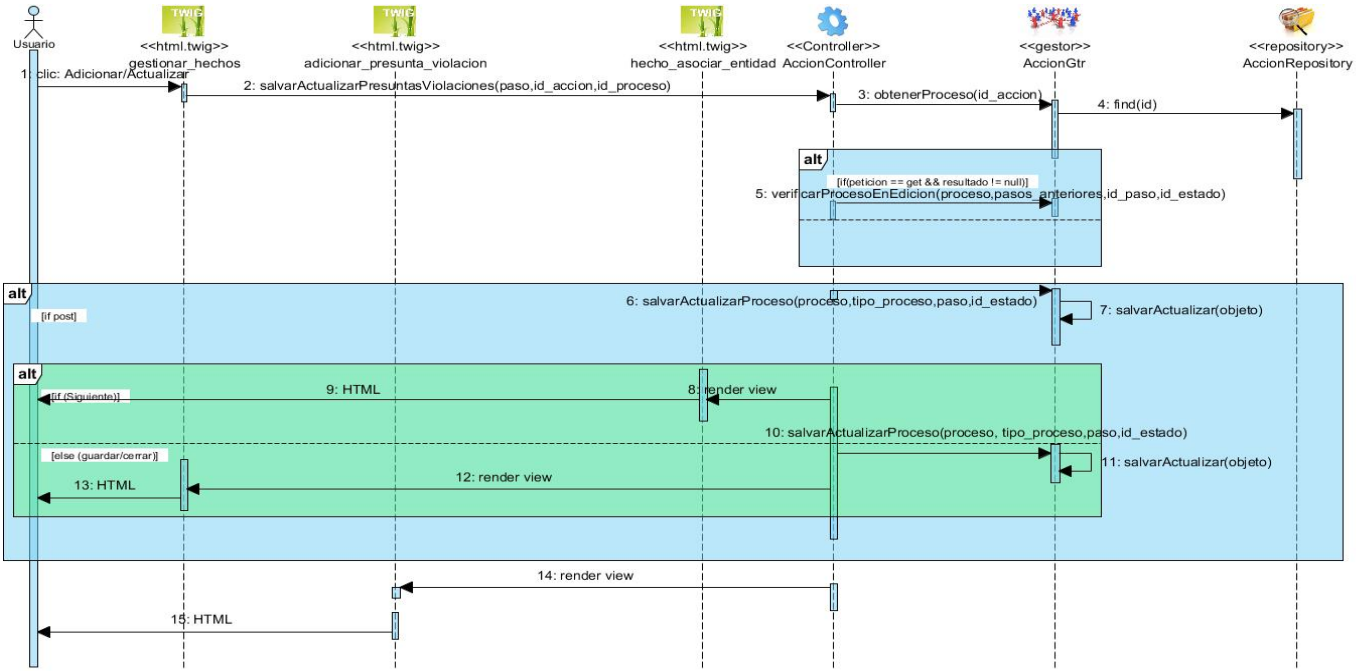


Figura 18 : Diagrama de secuencia de la funcionalidad Adicionar Presuntas Violaciones.

En la figura 18 el usuario selecciona la opción Adicionar o Actualizar en la página inicial donde se muestra el listado de todos los hechos. La acción activa la operación correspondiente en la clase controladora AccionEjecucionController, la cual verifica que el proceso se encuentre en edición y muestra el formulario para que se introduzcan los datos. Se muestra además en el diagrama las acciones que se ejecutan cuando se selecciona la opción “Siguiente”, pues se accede a la interfaz que asocia la entidad al proceso. En caso de seleccionar la opción “Guardar”, se accede a la interfaz relacionada con la asociación de las personas al proceso y se retorna a la página inicial.

2.4. Implementación del sistema

2.4.1. Diagrama de componentes

El modelo de implementación está comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se pueden encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. (45)

Un componente es una parte física de un sistema (módulo, base de datos, programa ejecutable). Muestra como el sistema está dividido en componentes y las dependencias entre ellos. Además proveen una vista arquitectónica de alto nivel del sistema y ayudan a los desarrolladores a visualizar el camino de la implementación. Permite tomar decisiones respecto a las tareas de implementación. (45)

A continuación se muestra el diagrama de componentes del sistema referente al módulo Verificación Fiscal.

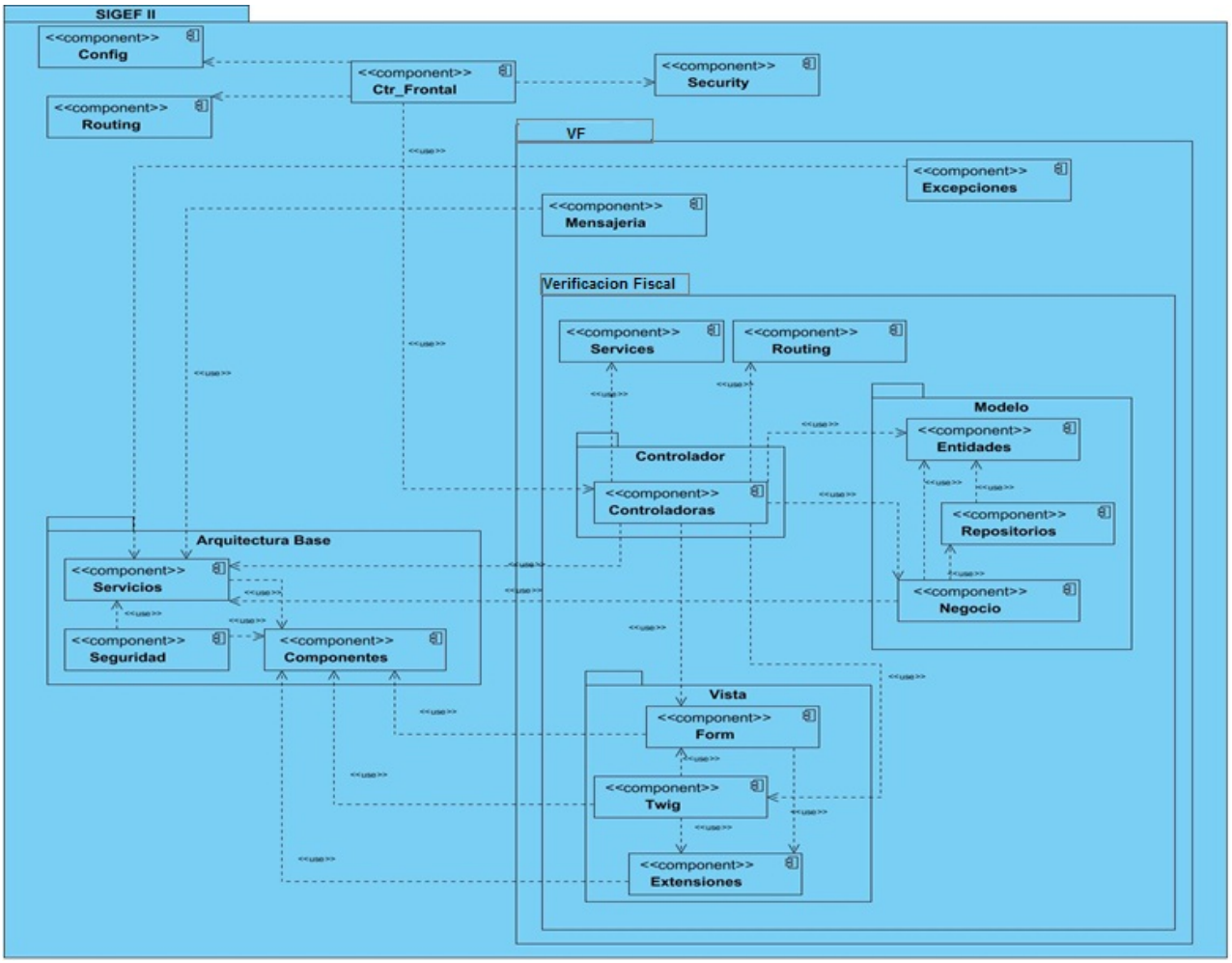


Figura 19: Diagrama de secuencia de la funcionalidad Adicionar/Actualizar Presuntas Violaciones.

El diagrama de componentes del módulo VF evidencia las interacciones entre la aplicación SIGEFII, el subsistema VF y el módulo Verificación Fiscal. SIGEF II incluye los componentes Ctr_Frontal, que es el controlador frontal que recibe cada petición realizada al sistema, Config (para las configuraciones), Routing (para las rutas de acceso), Security (para la seguridad). Dentro se encuentran el paquete Arquitectura Base (contiene los componentes con funcionalidades comunes para todos los módulos, como son: Servicios, Seguridad y Componentes) y el subsistema VF, donde se encuentran los componentes

Mensajería (para la gestión de mensajes informativos y de errores) y Excepciones (para el manejo de las excepciones generadas).

El módulo Verificación Fiscal comprendido en el subsistema VF está compuesto por tres paquetes fundamentales, Controlador (encargado de dar respuesta a las peticiones de los usuarios), Modelo (contiene las clases entidades del módulo y lo relacionado con la lógica de negocio), Vista (encargado de la construcción y manejo de las interfaces de usuario). El paquete Controlador con el componente Controladoras, para dar respuesta a las peticiones realizadas por los usuarios posee una relación con el componente Routing para el manejo de las rutas definidas para el módulo y el componente Services que ofrece los servicios públicos a los que se puede acceder desde cualquier parte del subsistema VF y donde se registran las clases gestoras para que se pueda trabajar con ellas desde las controladoras. El componente Controladoras se relaciona con el componente Negocio para gestionar la lógica del negocio del módulo y el manejo de las entidades, el componente Entidades es usado por el componente Repositorio que contiene las clases repositorios encargadas de realizar las consultas a la base de datos. El paquete Vista incluye los componentes Form (encargado del trabajo con los formularios), Twig (para las interfaces de usuarios y plantillas del módulo) y Extensiones (posee los javascript y css).

2.4.2. Estándares de codificación utilizados

La adopción de un estándar de codificación solo es viable si se sigue desde el principio hasta el final del proyecto de software. En el caso del módulo Verificación Fiscal, el estándar de codificación fue definido por la dirección del proyecto en sus inicios y se encuentra registrado en el documento *"Estándares de codificación para PHP"* del proyecto SIGEF II. A continuación se muestran algunas pautas utilizadas de dicho estándar que se presentan en el documento antes mencionado.

Cabecera del archivo:

- ✓ Siempre es importante que todos los archivos .php inicien con una cabecera específica que indican información sobre el mismo.

```
/**
 * @access public
 * @package SIGEFII.VF.VFBundle.Controller
 * @autor Kenia Lopez
 */
```

Figura 20: Cabecera de inicio de los archivos .php

- ✓ Cada archivo especificará el paquete que lo contiene y el uso de otras clases.

```
namespace VF\VFBundle\Controller;

use VF\VFBundle\Controller\VFController;
use Symfony\Component\HttpFoundation\Response;
```

Figura 21: Cabecera de inicio de los archivos .php

Comentarios en las funciones:

Todas las funciones deben tener un comentario, antes de su declaración, explicando que hacen. Ningún programador debe tener que analizar el código de una función para conocer su utilidad. Los comentarios se incluirán delante de cada función en forma de bloque (/* */) o cuando se estime necesario para aclaraciones dentro del código con la notación (//). A continuación se brinda un ejemplo del uso de los comentarios usados:

```
/*salvar actualizar el proceso presuntas violaciones*/
public function salvarActualizarPresuntasViolacionesAction ($paso, $id_accion, $id_proceso) {
```

Figura 22 : Comentarios en bloque

```
// obtiene el hecho o crea uno nuevo
$proceso = $this->getAccionGtr () ->crearHecho ($id_accion, $id_proceso);
```

Figura 23: Comentarios en una sola línea

Nombres de las clases:

Las clases serán colocadas en un archivo .php aislado, donde sólo se colocará el código de la clase. El nombre del archivo será el mismo de la clase y siempre empezará en mayúscula.

Los nombres de clases deben tener una sola palabra. Se escriben con la primera letra de cada palabra que lo compone en mayúscula, haciendo uso de la notación Camello, con la variante Upper Camel Case. Con excepción de los nombres de las páginas twig que serán en minúscula siempre y underscore "_", por ejemplo: adicionar_presunta_violacion.html.twig.

Las clases siguen las mismas reglas de las funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad.

Las clases formularios comienzan con el nombre del formulario según su función, seguido de la palabra Type (PresuntasViolacionesType.php).

Las clases gestoras de negocio comienzan con el nombre del gestor seguido de Gtr (*AccionGtr.php*).

```
class AccionController extends VFController {
```

Figura 24: Nombre de las clases

Nombre de las funciones:

- ✓ Comienzan con minúscula y continúan con mayúscula, haciendo uso de la notación Camello, con la variante *Lower Camel Case.*

```
public function salvarActualizarPresuntasViolacionesAction ($paso, $id_accion, $id_proceso)
```

Figura 25: Ejemplo de los nombres de las funciones

Nombres de variables:

- ✓ Los nombres deben ser descriptivos y concisos. No usar grandes frases, ni pequeñas abreviaturas. El identificador para las variables y los parámetros serán con letras en minúsculas y en caso de ser un nombre compuesto se divide cada palabra por un signo de underscore "_".

```
$peticion = $this->getRequest ();
```

Figura 26: Ejemplo de nombre de variables

2.5. Conclusiones parciales

Teniendo en cuenta el trabajo realizado en el presente capítulo, se arriba a las siguientes conclusiones:

- La realización de los diagramas de clases de diseño permitió modelar la estructura de la aplicación brindando los principales elementos para llevar a cabo la implementación del sistema.
- Con el objetivo de garantizar una visión general para la implementación de los procesos se realizó el diagrama de componentes.
- El uso de los estándares de codificación definidos en el proyecto, facilitó un mayor entendimiento para la implementación de los procesos.

CAPITULO 3: VALIDACION DE LA PROPUESTA DE SOLUCIÓN

3.1. Introducción

En el presente capítulo se lleva a cabo la validación del diseño y del sistema implementado. Permitirá constatar la calidad del resultado de la implementación, mediante los artefactos generados durante el flujo de trabajo de pruebas, exponiendo los resultados obtenidos por diferentes tipos de pruebas realizadas al sistema.

3.2. Validación del diseño

Para la validación del diseño de la funcionalidad “Adicionar presuntas violación de la legalidad” se utilizaron las métricas TOC y RC, explicadas en el capítulo 1. A continuación se muestra un ejemplo de la aplicación de las métricas a las clases correspondientes a la funcionalidad seleccionada.

Métrica TOC

A continuación se muestran los resultados obtenidos tras la aplicación de la métrica tamaño operacional de clase, donde:

R: responsabilidad

TC: tamaño de la clase

Re: reutilización de la clase

CP: cantidad de procedimientos

CA: cantidad de atributos de la clase

CO: cantidad de operaciones de la clase

CI: complejidad de implementación de la clase

No	Clases	CA	CO	CP	TC	R	CI	Re
1	*Hecho*	12	24	38	Grande	Baja	Baja	Alta
2	*Accion*	14	28	42	Grande	Media	Media	Media
3	*Proceso*	8	16	24	Medio	Baja	Baja	Alta
4	*PersonaProceso*	3	6	9	Pequeño	Baja	Baja	Alta
5	*Violacion*	13	26	39	Grande	Baja	Baja	Alta
6	*AccionRepository*	0	1	1	Pequeño	Baja	Baja	Alta
7	*AccionController*	0	45	45	Grande	Media	Media	Media
8	*ProcesoController*	0	36	36	Grande	Baja	Baja	Alta
9	*AccionGtr*	1	85	86	Grande	Alta	Alta	Baja
10	*ProcesoGtr*	6	74	80	Grande	Media	Media	Media

Tabla 8: Resultados generales de la aplicación de la métrica TOC a las clases de la funcionalidad seleccionada.

Se trabajó con un total de 10 clases para un promedio de 40 de cantidad de procedimientos, obteniéndose como resultados los datos que a continuación se muestran:

Cantidad de clases: 10	Baja	Media	Alta
Responsabilidad	60%	30%	10%
Complejidad de implementación	60%	30%	10%
Reutilización	10%	30%	60%

Tabla 9: Resultados generales de la aplicación de la métrica TOC a las clases de la funcionalidad seleccionada.

El resultado de aplicar la métrica TOC demuestra que de un total de 10 clases pertenecientes a la funcionalidad “Adicionar presunta violación de la legalidad”, 6 poseen baja responsabilidad, lo que representa un 60% del total. Además, se obtuvieron seis clases con baja complejidad de implementación. Se obtuvo una clase con baja reutilización representando el 10%. Los resultados demuestran que dicha funcionalidad posee una elevada reutilización, baja complejidad y responsabilidad en el diseño propuesto.

Métrica RC

A continuación se muestran los resultados obtenidos para cada atributo de calidad evaluado, tras la aplicación de la métrica relaciones entre clases:

Donde:

CR: cantidad de relaciones de uso.	CM: complejidad de mantenimiento.
A: acoplamiento entre clases.	CP: cantidad de pruebas.

No	Clases	CR	A	CM	CP
1	*Hecho*	3	Alto	Media	Media
2	*Accion*	3	Alto	Media	Media
3	*Proceso*	4	Alto	Media	Media
4	*PersonaProceso*	2	Medio	Baja	Baja
5	*Violacion*	4	Alto	Media	Media
6	*AccionRepository*	1	Bajo	Baja	Baja
7	*AccionController*	1	Bajo	Baja	Baja
8	*ProcesoController*	1	Bajo	Baja	Baja
9	*AccionGtr*	1	Bajo	Baja	Baja
10	*ProcesoGtr*	1	Bajo	Baja	Baja

Tabla 10: Resultados de la aplicación de la métrica RC a las clases de la funcionalidad seleccionada.

Se obtuvo un promedio de 2.1 asociaciones entre clases. A continuación se muestran los datos que resultaron tras la aplicación de la métrica:

Cantidad de clases: 10	Baja	Media	Alta
Acoplamiento	50%	10%	40%
Complejidad de mantenimiento	60%	40%	0%
Cantidad de pruebas	60%	40%	0%

Tabla 11: Resultados generales de la aplicación de la métrica RC a las clases de la funcionalidad seleccionada.

Los resultados obtenidos luego de aplicar la métrica RC, demuestran que la mayoría de las clases que componen la funcionalidad “Adicionar presunta violación de la legalidad”, representadas por un 50 % poseen pocas relaciones de uso, por lo que tienen un bajo acoplamiento, además poseen un bajo grado de complejidad de mantenimiento y bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones y pruebas al software.

3.3. Validación del sistema

La validación del sistema es el proceso de revisión que verifica que el sistema de software producido cumple con todos sus requerimientos, de esta manera se comprueba que lo que se ha especificado es lo que el usuario realmente quiere. Este proceso se realiza mediante pruebas realizadas al software explicadas anteriormente el capítulo 1.

3.3.1. Prueba de caja blanca

Para desarrollar las pruebas de caja blanca se empleó la técnica camino básico para los métodos más complejos. La misma permitió obtener un resultado de la complejidad lógica para el diseño de los casos de pruebas y usar este resultado como guía para la definición de un conjunto básico de caminos de ejecución.

El método fundamental que se toma como ejemplo es *salvarActualizarPresuntas ViolacionesAction* que se encuentra en la clase *AccionController*. (Ver Figura 27)

```
$peticion = $this->getRequest();
$proceso = $this->getAccionGtr()->crearHecho($id_accion,$id_proceso);
$usuario = $this->getUser();
$resultado = $this->verificarProcesoEnEdicion($proceso, $this->listaPasosAnteriores($paso), $paso,
ConfigUtilVF::estado_edicion, $this->generateUrl('vf_gestionar_presuntas_violaciones', array('paso' => ConfigUtilVF::Iniciar_Hecho)));
if (!is_null($resultado)) {
    return $resultado;
}
$frm = $this->createForm(new PresuntasViolacionesType(),$proceso);
if ($peticion->getMethod() == 'POST') {
    $frm->bind($peticion);
    $proceso->setFechaCreacion(new \DateTime());
    if ($frm->isValid()) {
        if ($peticion->get(ConfigUtilVF::btn_siguiente)) {
            $proceso = $this->getAccionGtr()->salvarActualizarProceso($proceso, ConfigUtilVF:: Hecho_VF, $paso,
            ConfigUtilVF::estado_edicion, $usuario);
            return $this->redirect($this->generateUrl('vf_asociar_entidad_proceso', array('paso' => $paso,'id_accion' => $id_accion,
            'id_proceso' => $proceso->getIdProceso())));
        }
        else if ($peticion->get(ConfigUtilVF::btn_guardar)) {
            $proceso = $this->getAccionGtr()->salvarActualizarProceso($proceso, ConfigUtilVF::Hecho_VF, $paso,
            ConfigUtilVF::estado_pendiente, $usuario);
            return $this->redirect($this->generateUrl('vf_gestionar_presuntas_violaciones', array('paso' => $paso,'id_proceso' => $id_accion)));
        }
    }
}
return $this->render('VFBundle:Violacion:adicionar_presunta_violacion.html.twig', array(
    'frm' => $frm->createView(),
    'paso' => $paso,
    'id_accion' => $id_accion,
    'id_proceso' => $id_proceso));
```

Figura 27: Código fuente del método salvarActualizarPresuntasViolacionesAction

A continuación se muestra el grafo de flujo generado a partir de la aplicación de la técnica camino básico.

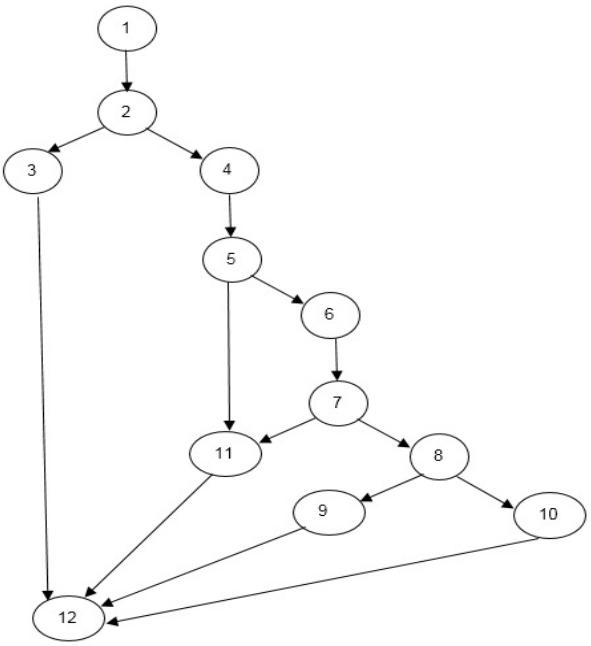


Figura 28 : Grafo de flujo del método salvarActualizarPresuntasViolacionesAction

Después de haber realizado el gafo de flujo se procede a calcular la complejidad ciclomática:

1. V (G)= 5
2. V (G)=A-N+2=15-12+2=5
3. V (G)=P+1=4+1=5

El valor de V (G) fue igual al número de caminos linealmente independientes de la estructura de control del programa, por lo que se definieron los siguientes 5 caminos:

Camino1: 1-2-3-12

Camino2: 1-2-4-5-11-12

Camino3: 1-2-4-5-6-7-11-12

Camino4: 1-2-4-5-6-7-8-9-12

Camino5: 1-2-4-5-6-7-8-10-12

Después de haber confeccionado el grafo de flujo y los caminos a recorrer, se realizan los casos de prueba que forzaron la ejecución de cada uno de esos caminos. A continuación se muestran los resultados de la aplicación de la prueba sobre el método salvarActualizarPresuntasViolacionesAction.

Tabla 12: Caso de prueba del camino básico 5.

Entrada	Se necesita que la petición se haya realizado por el método POST, los datos del formulario deben ser válidos, el botón seleccionado debe ser Guardar y Cerrar.
Resultados esperados	Los datos de la presunta violación se salvan en la base de datos.
Condiciones	$resultado=null, $peticion->getMethod() = 'POST', $frm->isValid()=true, isset($tipo_boton) =false, count($errores)=0

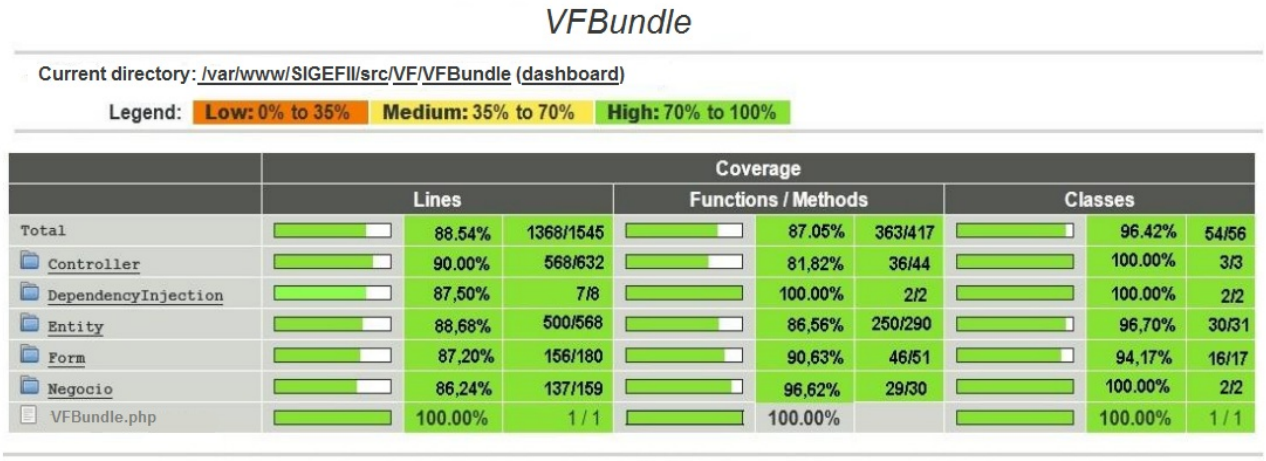
Como parte de las pruebas de caja blanca aplicadas se hizo empleo del framework PHPUnit, el cual es utilizado para realizar pruebas unitarias y analizar los resultados obtenidos en ellas. Se llevaron a cabo 5 pruebas correspondientes a la funcionalidad propuesta, de las cuales para una primera iteración 1 se generaron errores, mientras que el resto resultaron satisfactorias. Los errores generados fueron debido a irregularidades que presentaba el código fuente como se muestra en la figura 29, donde las líneas de código en color verde fueron ejecutadas de forma correcta, las líneas en naranja no fueron ejecutadas y las líneas en gris representan errores. El error encontrado en el código fue resuelto añadiéndole el parámetro que le faltaba, en este caso, el paso que es necesario para que la ruta funcione correctamente. En una segunda iteración no se detectaron errores.

```
public function salvarActualizarPresuntasViolacionesAction($paso,$id_accion,$id_proceso){
    $peticion = $this->getRequest();
    $proceso = $this->getAccionGtr()->crearHecho($id_accion,$id_proceso);
    $usuario = $this->getUser();
    $resultado = $this->verificarProcesoEnEdicion($proceso, $this->listaPasosAnteriores($paso), $paso, ConfigUtilVF::estado_e
        $this->generateUrl('vf_gestionar_presuntas_violaciones', array('paso' => ConfigUtilVF::Iniciar_Hecho)));
    if (!is_null($resultado)) {
        return $resultado;
    }
    $frm = $this->createForm(new PresuntasViolacionesType(),$proceso);
    if ($peticion->getMethod() == 'POST') {
        $frm->bind($peticion);
        $proceso->setFechaCreacion(new \DateTime());
        if ($frm->isValid()) {
            if ($peticion->get(ConfigUtilVF::btn_siguiente)) {
                $proceso = $this->getAccionGtr()->salvarActualizarProceso($proceso, ConfigUtilVF:: Hecho_VF, $paso, ConfigUti
                return $this->redirect($this->generateUrl('vf_asociar_entidad_proceso', array('paso' =>      ,'id_accion' =>
                    'id_proceso' => $proceso->getIdProceso())));
            }
            else if ($peticion->get(ConfigUtilVF::btn_guardar)) {
                $proceso = $this->getAccionGtr()->salvarActualizarProceso($proceso, ConfigUtilVF::Hecho_VF, $paso, ConfigUtil
                return $this->redirect($this->generateUrl('vf_gestionar_presuntas_violaciones', array('paso' => $paso,'id_pro
            }
        }
    }
    return $this->render('VFBundle:Violacion:adicionar_presunta_violacion.html.twig', array(
        'frm' => $frm->createView(),
        'paso' => $paso,
        'id_accion' => $id_accion,
        'id_proceso' => $id_proceso));
```

Figura 29: Errores que detectó PHPUnit en la primera iteración

Resultados obtenidos

Se realizó la prueba de caja blanca a los procesos detección de violaciones y responsables a través de la herramienta PHPUnit; en una primera iteración se descubrieron 18 fallos que posteriormente fueron analizados y resueltos. Seguidamente se muestran los resultados de la aplicación de dichas pruebas.



VFBundle

Current directory: /var/www/SIGEFII/src/VF/VFBundle (dashboard)

Legend: Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

	Coverage								
	Lines			Functions / Methods			Classes		
Total	88.54%	1368/1545		87.05%	363/417		96.42%	54/56	
Controller	90.00%	568/632		81,82%	36/44		100.00%	3/3	
DependencyInjection	87,50%	7/8		100.00%	2/2		100.00%	2/2	
Entity	88,68%	500/568		86,56%	250/290		96,70%	30/31	
Form	87,20%	156/180		90,63%	46/51		94,17%	16/17	
Negocio	86,24%	137/159		96,62%	29/30		100.00%	2/2	
VFBundle.php	100.00%	1 / 1		100.00%			100.00%	1 / 1	

Figura 30: Resultados de las pruebas de caja blanca.

Después de aplicadas las pruebas a los procesos de detección de violaciones y responsables se obtuvo como resultado un 88.54% de las líneas de códigos ejecutadas, un 87.05% de las funcionalidades ejecutadas y un 96.42% de las clases ejecutadas. La prueba de la caja blanca demostró que el estado real del software coincide con el esperado, comprobándose a través de los casos de prueba desarrollados en la herramienta PHPUnit donde la ejecución de las condiciones tuvo resultados satisfactorios.

3.3.2. Prueba de caja negra

Para la realización de las pruebas de caja negra se aplicó la técnica partición de equivalencia. A continuación se muestra un ejemplo de caso de prueba del requisito Adicionar presunta violación de la legalidad.

Escenario	Descripción	Fecha de ocurrencia	Fecha de detección	Síntesis del hecho	Persona que la detecta	Respuesta del sistema	Flujo central
EC1. Adicionar o Actualizar presuntas violaciones con los datos	**Permite adicionar o actualizar una presunta Violación exitosamente.**	V 09/02/2009	V 09/02/2012	V Desvío de recursos en la empresa del pan.	V Fiscal actuante	**Adiciona o actualiza la presunta violación.**	**1-Selecciona la opción Iniciar hechos del menú principal del proceso.**

correctos.							
EC2. Adicionar o Actualizar presuntas violaciones con los campos vacíos.	**No permite adicionar o actualizar una presunta violación porque faltan campos por llenar.**	N/A	V	V	V	**El sistema debe mostrar un mensaje indicando que existen campos obligatorios vacíos, al seleccionar la opción siguiente.**	**2-Selecciona la acción del listado de acciones para mostrar los hechos asociados dicha acción.** **Selecciona la opción Adicionar o Actualizar.** **Muestra una pantalla con los datos a introducir o con los valores a editar.** **Salva el hecho y asocia la entidad y las personas al proceso.**
		Vacío	09/02/2012	Desvío de recursos en la empresa del pan.	Fiscal actuante		
		V	N/A	V	V		
		09/02/2009	Vacío	Desvío de recursos en la empresa del pan.	Fiscal actuante		
		V	V	N/A	V		
		09/02/2009	09/02/2012	Vacío	Fiscal actuante		
		V	V	V	N/A		
		09/02/2009	09/02/2012	Desvío de recursos en la empresa del pan.	Vacío		
EC3. Adicionar o Actualizar presuntas violaciones con los datos incorrectos.	**Permite adicionar o actualizar una presunta violación con datos incorrectos.**	I	V	V	V	**Debe mostrar un mensaje indicando que la Fecha de ocurrencia es menor que la fecha en que se detectó la violación.**	
		25/02/2009	22/02/2009	Desvío de recursos en la empresa del pan.	Fiscal actuante		

Tabla 12: Caso de prueba Adicionar presuntas violaciones

Al concluir las pruebas de caja negra para la funcionalidad Adicionar presunta violación de la legalidad, en una primera iteración se detectaron un total de cinco no conformidades, las cuales fueron resueltas. Al aplicar una segunda iteración de estas pruebas para dicha funcionalidad se obtuvieron resultados satisfactorios.

Resultados obtenidos

Se realizaron pruebas de caja negra a los 54 requisitos funcionales pertenecientes a los procesos de detección de violaciones y responsables. De estos, en la primera iteración se obtuvieron 15 no conformidades. En una segunda iteración se corrigieron las no conformidades y resultó un total de 5 errores; luego para una tercera iteración con un total de 54 casos de prueba se obtuvo un resultado correcto.

3.4. Validación de la propuesta solución

En un espacio que se tuvo con los fiscales encargados del área VF, se sostuvo un intercambio en el cual se obtuvieron los tiempos asociados a la realización de los procesos violaciones y responsables. Posteriormente se realizó una prueba piloto donde interactuó uno de los fiscales con el sistema para conocer el comportamiento de las variables celeridad y control de los procesos.

A partir de estos datos, se realizó una comparación en cuanto al comportamiento de las variables de la investigación: control y celeridad. Los resultados obtenidos fueron los siguientes:

Dimensión de la variable	Antes	Después
Acceso a la información	No se conocen los detalles del proceso si no se tiene la Resolución de la acción. [3]	Si se cuenta con los permisos necesarios, se puede acceder a la información de la acción desde cualquier instancia.

Tabla 13: Validación para la variable control

Dimensión de la variable	Antes	Después	Indicadores
Documentos generados	Aproximadamente 2 horas.	Aproximadamente 15 minutos.	Tiempo de creación.

Tabla 14: Validación para la variable celeridad

[3] Resolución de la acción: Es el documento oficial que marca el fin de una investigación o verificación fiscal. Contiene toda la información concerniente los resultados de la acción ejecutada.

Las tablas mostradas anteriormente demuestran que con el desarrollo de los procesos detección de violaciones y responsables se mejora la gestión de la información, contribuyendo al control y celeridad de su ejecución, en cuanto a:

- control del acceso a la información de los usuarios.
- disponibilidad de la información.
- reducción del tiempo de generación de documentos.

3.5. Conclusiones parciales

Teniendo en cuenta el análisis realizado en el presente capítulo, se arriba a las siguientes conclusiones:

- Al aplicar las pruebas de caja blanca y caja negra se demostró el correcto funcionamiento de la aplicación implementada y que la misma cumple con los requisitos definidos por el cliente.
- Se logró validar la propuesta de solución frente al problema planteado, demostrando que existe un mayor control sobre los procesos y se disminuye el tiempo de búsqueda de información.

CONCLUSIONES GENERALES

Luego de realizar el presente trabajo se concluye que:

- El estudio de la metodología, arquitectura y requisitos permitió realizar el diseño de los procesos de violaciones y responsables del módulo Verificación Fiscal del proyecto SIGEF II para su posterior implementación.
- La solución desarrollada fue validada demostrando que las funcionalidades del sistema son correctas, que las entradas se aceptan, se producen salidas satisfactorias y los componentes funcionan de forma adecuada.
- Con la propuesta de solución se mejora la gestión de la información de los procesos detección de violaciones y responsables del módulo VF, garantizando control y celeridad en su ejecución.

RECOMENDACIONES

- Desplegar los procesos implementados para mejorar, de manera general, la gestión de estos en las diferentes fiscalías del país.
- Implementar los restantes procesos del módulo Verificación Fiscal de forma que se obtenga un producto completamente funcional.

BIBLIOGRAFÍA

1. Conceptos Básicos de Computación. [En línea] [Citado el: 26 de Enero de 2013.] http://dcb.fi-c.unam.mx/users/miguelegc/tutoriales/tutorialcd/mpct_cmpt_scd_1.htm.

2. EVA. *EVA.* [En línea] 3 de diciembre de 2012. http://eva.uci.cu/file.php/103/Bibliografia_semana_16/Politicas_TIC_para_el_desarrollo.pdf.

3. *Articulo 127 Constitución de la República.*

4. Calderio, Blas Roca. *Ley de Procedimiento Civil, Administrativo y Laboral.* 1977.

5. Buenas Tareas. *Buenas Tareas.* [En línea] [Citado el: 29 de junio de 2014.] http://www.buenastareas.com/ensayos/Principios-De-Legalidad/978471.html.

6. Piñero Pérez, Pedro. *Metodologías Ágiles y Robustas.* 2009.

7. Ecured. *Ecured.* [En línea] [Citado el: 31 de marzo de 2014.] http://www.ecured.cu/index.php/Metodologias_de_desarrollo_de_Software.

8. Booch, Greddy, Jacobson, Ivar y Rumbaugh, James. *El proceso unificado de desarrollo de software.* 2000.

9. Figueroa Machado, Yenier. *Proyecto Técnico del Sistema de Informatización de las Fiscalías fase II.*

10. PRESSMAN. *Ingeniería del Software: Un Enfoque Práctico.* 2005.

11. Calisoft. . *Calisoft. .* [En línea] [Citado el: 21 de mayo de 2014.] http://calisoft.uci.cu/index.php/proceso-de-mejora..

12. Calisoft. *Programa de Mejora. Introducción a la Administración de Requisitos. Tema I. .*

13. Figueroa, Yenier, Fuentes, Hector y Delgado, Manuel. *Propuesta de arquitectura de software para proyectos de gestión sobre plataformas web.* 2014.

14. *Linea Base de Arquitectura SIGEF II.* Habana : s.n., 2014.

15. Modelo de implementacion. *Modelo de implementacion.* [En línea] 1 de junio de 2013. [Citado el: 14 de mayo de 2014.] http://ithleovi.blogspot.com/2013/06/unidad-5-modelo-deimplementacion-el.html.

16. EVA. *EVA.* [En línea] http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Arquitectura_de_Software/Arquitecturas_de_software._Guias_de_estudio.pdf.

17. Eguiluz, Javier. *Desarrollo web ágil con Simfony 2.* 2011.

18. Symfony para Todos. *Symfony para Todos.* [En línea] [Citado el: 19 de febrero de 2014.] http://symfony.cubava.cu/introduccion/symfony-2/.

19. Sobre PostgreSQL. *Sobre PostgreSQL.* [En línea] [Citado el: 19 de febrero de 2014.] http://www.postgresql.org.es/sobre_postgresql.

20. PostgreSQL. *PostgreSQL.* [En línea] [Citado el: 19 de febrero de 2014.] http://postgresql-dbms.blogspot.com/p/limitaciones-puntos-de-recuperacion.html.

21. Ecured. *Ecured.* [En línea] [Citado el: 3 de marzo de 2014.] http://www.ecured.cu/index.php/PHP#Aplicaciones_desarrolladas_con_PHP.

22. Ecured. *Ecured.* [En línea] [Citado el: 3 de marzo de 2014.] lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla. .

23. LIBROSWEB. *LIBROSWEB.* [En línea] [Citado el: 3 de marzo de 2014.] http://librosweb.es/javascript/capitulo_1.html.

24. LibrosWeb. *LibrosWeb.* [En línea] [Citado el: 3 de marzo de 2014.] http://librosweb.es/css/capitulo_1.html.

25. Web Técnica de TIC. *Web Técnica de TIC.* [En línea] [Citado el: 3 de marzo de 2014.] http://www.tic2.org/WebTecnica/Programacion/CSS/CSSDocEstructura/CSSDocEstructura.htm#IDAUGOY.

26. JL Design. *JL Design.* [En línea] [Citado el: 3 de marzo de 2014.] http://www.jldesign.com.mx/que-es-html-5-y-cuales-son-sus-nuevas-caracteristicas/.

27. software.ar.com. *software.ar.com.* [En línea] [Citado el: 9 de diciembre de 2013.] http://www.software.com.ar/visual-paradigm-para-uml.html.

28. Sitio de descargas de software. *Sitio de descargas de software.* [En línea] [Citado el: 9 de diciembre de 2013.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

29. EcuRed. *EcuRed.* [En línea] [Citado el: 9 de diciembre de 2013.] http://www.ecured.cu/index.php/CASE#Visual_Paradigm.

30. GUÍA DOCUMENTADA PARA UBUNTU. *GUÍA DOCUMENTADA PARA UBUNTU.* [En línea] [Citado el: 19 de febrero de 2014.] http://www.guia-ubuntu.com/index.php/PgAdmin_III.

31. Recursostic. *Recursostic.* [En línea] [Citado el: 17 de marzo de 2014.] http://recursostic.educacion.es/observatorio/web/es/software/software-general/548-luis-garcia..

32. fotonostra. *fotonostra.* [En línea] [Citado el: 13 de marzo de 2014.] http://www.fotonostra.com/grafico/definiciondiseno.htm.

33. EVA. *EVA.* [En línea] [Citado el: 14 de marzo de 2014.] http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Diseno_de_software/Patrones_de_Diseno_Art._2.pdf.

34. EVA. *EVA.* [En línea] [Citado el: 14 de marzo de 2014.] http://eva.uci.cu/file.php/158/Documentos/Recursos_bibliograficos/Libros_y_articulos_UD_1/Arquitectura_de_Software/Arquitecturas_de_software._Guias_de_estudio.pdf.

35. Ecured. *Ecured.* [En línea] [Citado el: 4 de marzo de 2014.] http://www.ecured.cu/index.php/M%C3%A9trica_de_dise%C3%B1o.

36. Ecured. *Ecured.* [En línea] [Citado el: 14 de mayo de 2014.] http://www.ecured.cu/index.php/M%C3%A9trica_de_dise%C3%B1o#Atributos_de_calidad.

37. ITESCAM. *ITESCAM.* [En línea] [Citado el: 4 de marzo de 2014.] ww.itescam.edu.mx/principal/sylabus/fpdb/recursos/r21722.PDF.

38. Pruebas de Software. *Pruebas de Software.* [En línea] [Citado el: 6 de marzo de 2014.] http://pruebasdesoftware.com/laspruebasdesoftware.htm.

39. Tecnología y Synergix. *Tecnología y Synergix.* [En línea] [Citado el: 17 de marzo de 2014.] http://synergix.wordpress.com/2008/03/15/definimos-pruebas-de-unidad-como.

40. Ingeniero de Gestion. *Ingeniero de Gestion.* [En línea] [Citado el: 6 de marzo de 2014.] http://ingenierogestion.blogspot.com/2009/06/pruebas-de-caja-negra-y-caja-blanca.html.

41. Ecured. *Ecured.* [En línea] [Citado el: 6 de mayo de 2014.] http://www.ecured.cu/index.php/Pruebas_de_caja_negra.

42. Ecured. *Ecured.* [En línea] [Citado el: 28 de marzo de 2014.] http://www.ecured.cu/index.php/SeqProMaq_v4.0.1#Requerimientos_funcionales.

43. *Recommended Practice for Architectural Description of Software-Intensive Systems.* 2000.

44. Jacobson, Ivar, Booch, Greddy y Rumbaugh, James. *El proceso unificado de desarrollo de software. Traducción al español.* Madrid : Pearson Educacion S.A, 2000.

45. MSDN. [En línea] http://msdn.microsoft.com/es-es/library/bb972240.aspx..

46. Boyarynov, Eugene. *Symfony2 The Book.* 2011.

47. Sonda. *Sonda.* [En línea] [Citado el: 18 de marzo de 2014.] http://www.sonda.com/casos/12.

48. *Pliego de prescripciones técnicas para la contratación de servicios de mantenimiento de las aplicaciones del área de Fiscalías de la Subdirección General de Nuevas Tecnologías de la Justicia.* España : s.n., 2009.

49. *Rational Unified Process.* 2000.

50. Carlos Javier Perez Escobar. Qué significa CMMI. *Qué significa CMMI.* [En línea] 17 de septiembre de 2013. [Citado el: 20 de mayo de 2014.] http://asprotech.blogspot.com/2013/09/categorias-de-requisitos-no-funcionales.html.

GLOSARIO DE TÉRMINOS

BPM: Es una tecnología para el modelado de los procesos del negocio. Constituye un gran avance, un nuevo paradigma en cuanto a flexibilidad, gestión, control de información y datos; es el resultado de la combinación de avances técnicos con métodos y prácticas establecidos de un modelo empresarial centrado en el proceso. La tecnología BPM incluye todo lo que se necesita a la hora de diseñar, representar, analizar y controlar los procesos de negocio operacionales.

Entidades: organización administrativa, comercial, económica, productiva y de servicios de carácter estatal, cooperativa, privada o mixta, residentes en el territorio nacional; así como las organizaciones sociales y de masas del país.

Framework: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

HTML: Lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Namespace: es una colección de nombres, identificados por un URI[4], que se utiliza en los documentos XML para identificar los nombres de los elementos y atributos. (46)

Proceso: el proceso judicial se iniciará con la presentación de la demanda verbal o escrita

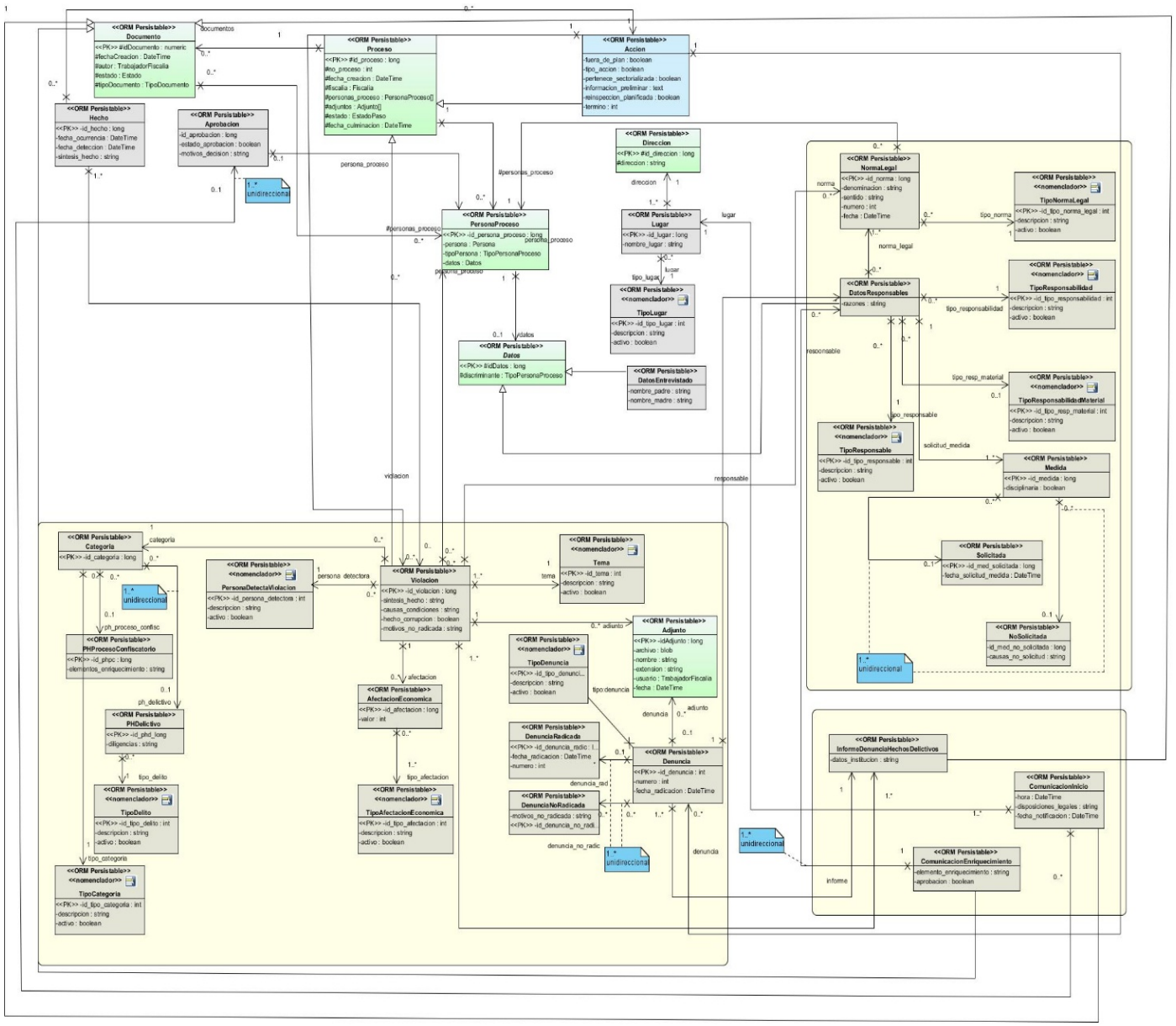
Sistema: Colección de componentes interrelacionados que trabajan conjuntamente para cumplir algún objetivos.

.

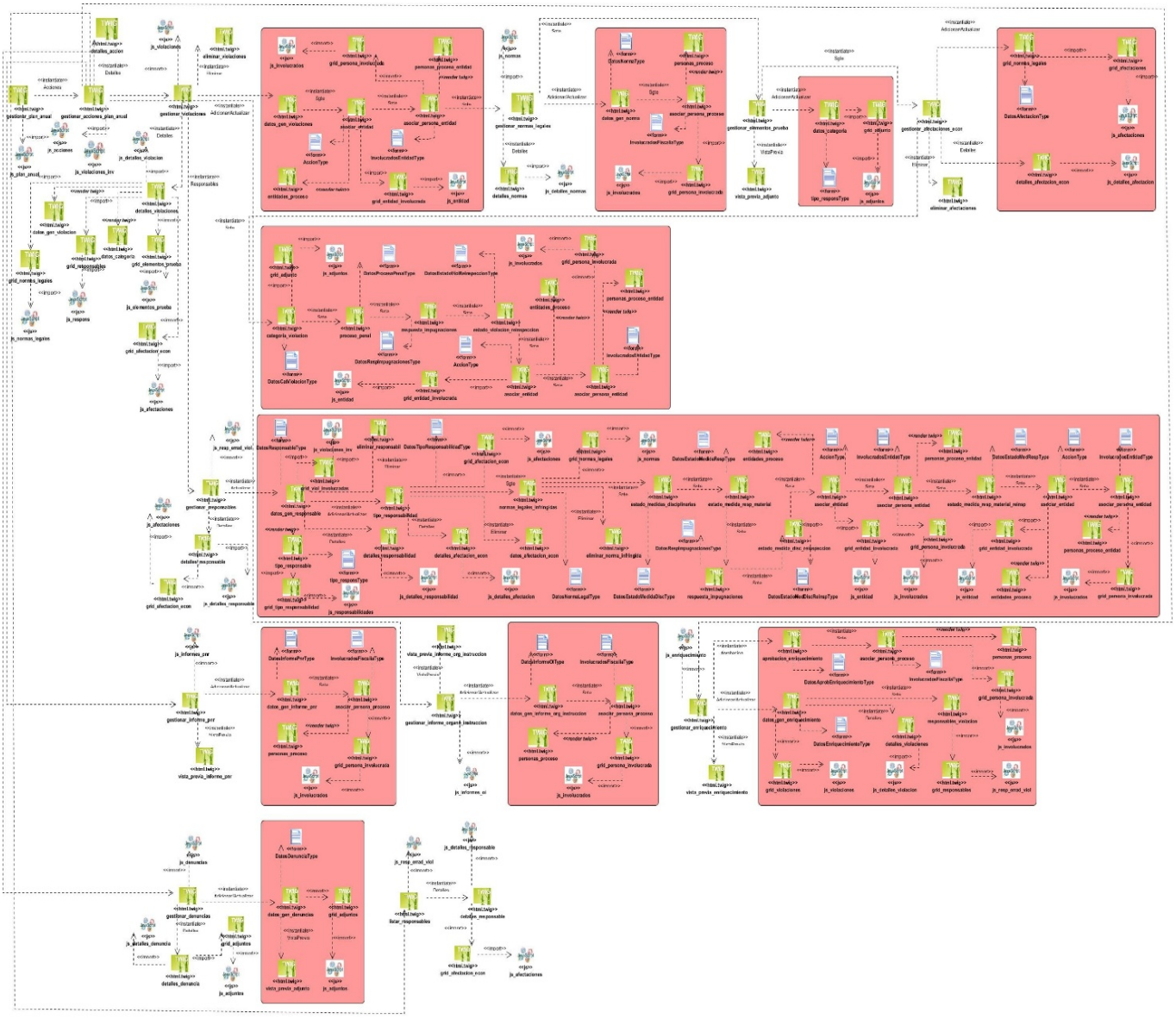
[4] **URI:** Identificador de Recurso Uniforme: Son cadenas que funcionan como identificadores globales que hacen referencia a recursos en la web, tales como documentos, imágenes, etc.

ANEXOS

Anexo # 1: Diagrama de clases entidades de los procesos detección de violaciones y responsables del módulo Verificación Fiscal del SIGEF II.

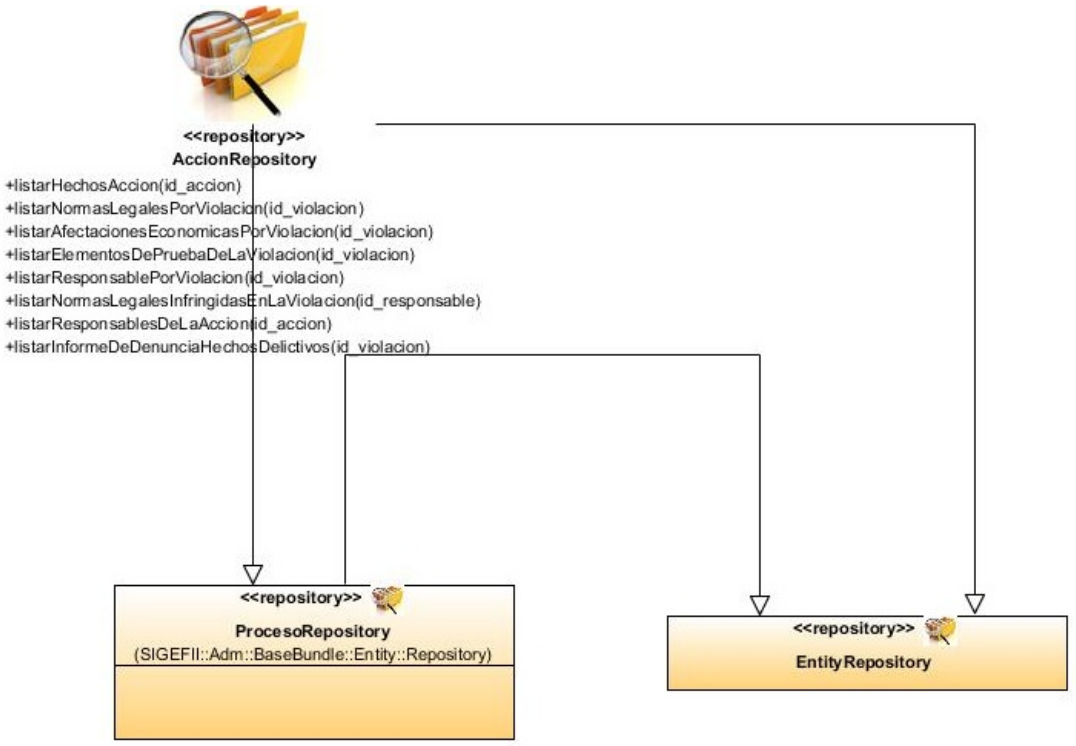


Anexo # 2: Diagrama de clases vistas de los procesos detección de violaciones y responsables del módulo Verificación Fiscal del SIGEF II.



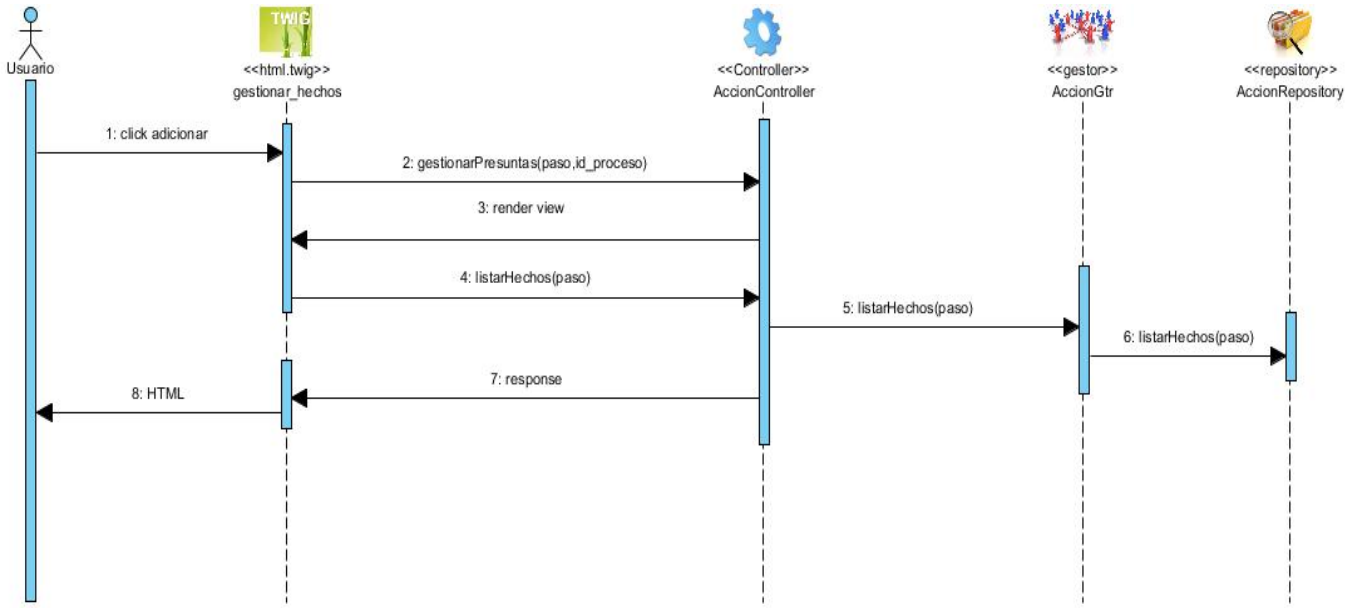
Anexo # 3: Diagrama de clases controladoras y gestoras de los procesos detección de violaciones y responsables del módulo Verificación Fiscal del SIGEF II.

Anexo # 4: Diagrama de clases repositorios de los procesos detección de violaciones y responsables del módulo Verificación Fiscal del SIGEF II.

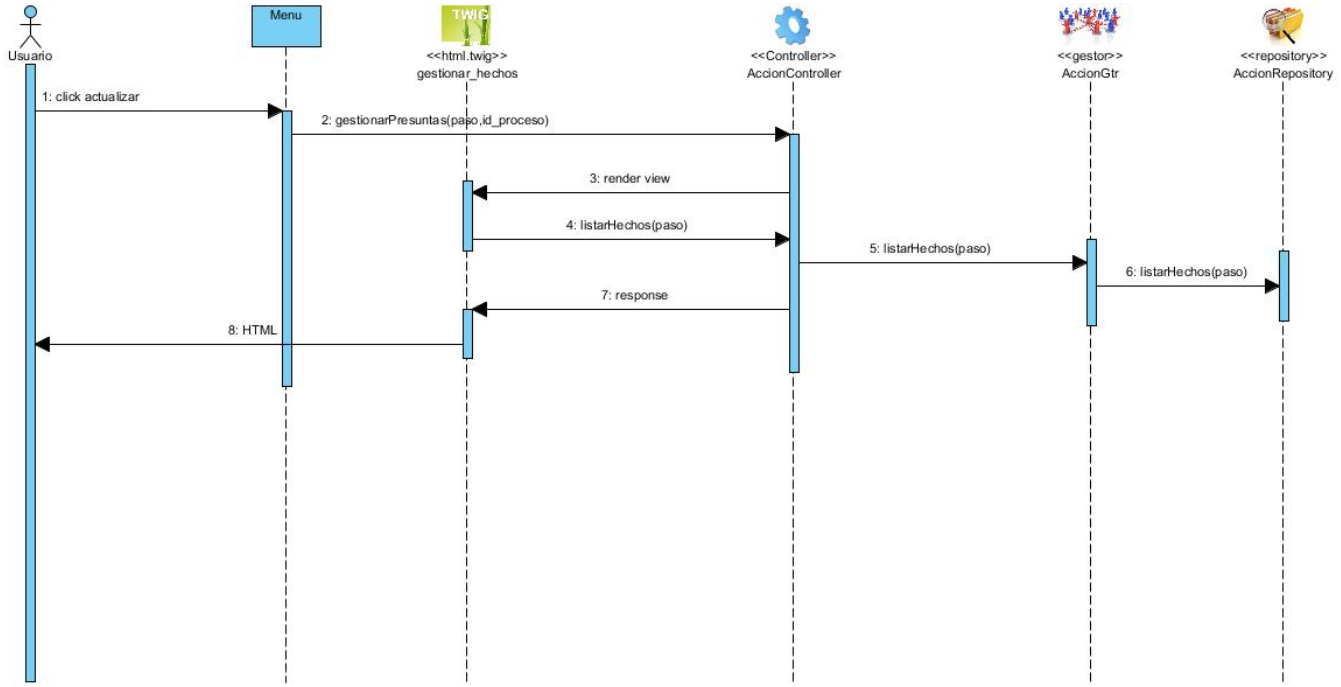


Anexo # 5: Diagramas de secuencia de la funcionalidad Adicionar presunta violación de la legalidad del módulo Verificación Fiscal del SIGEF II.

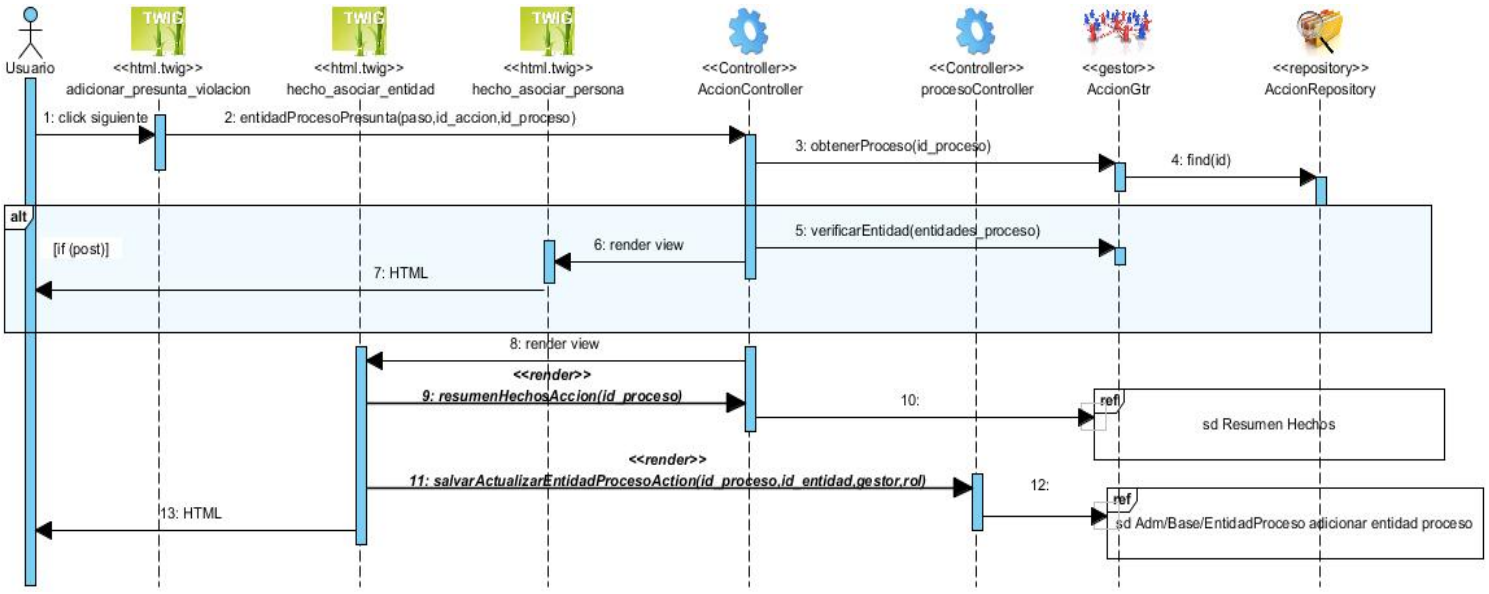
Listar acciones



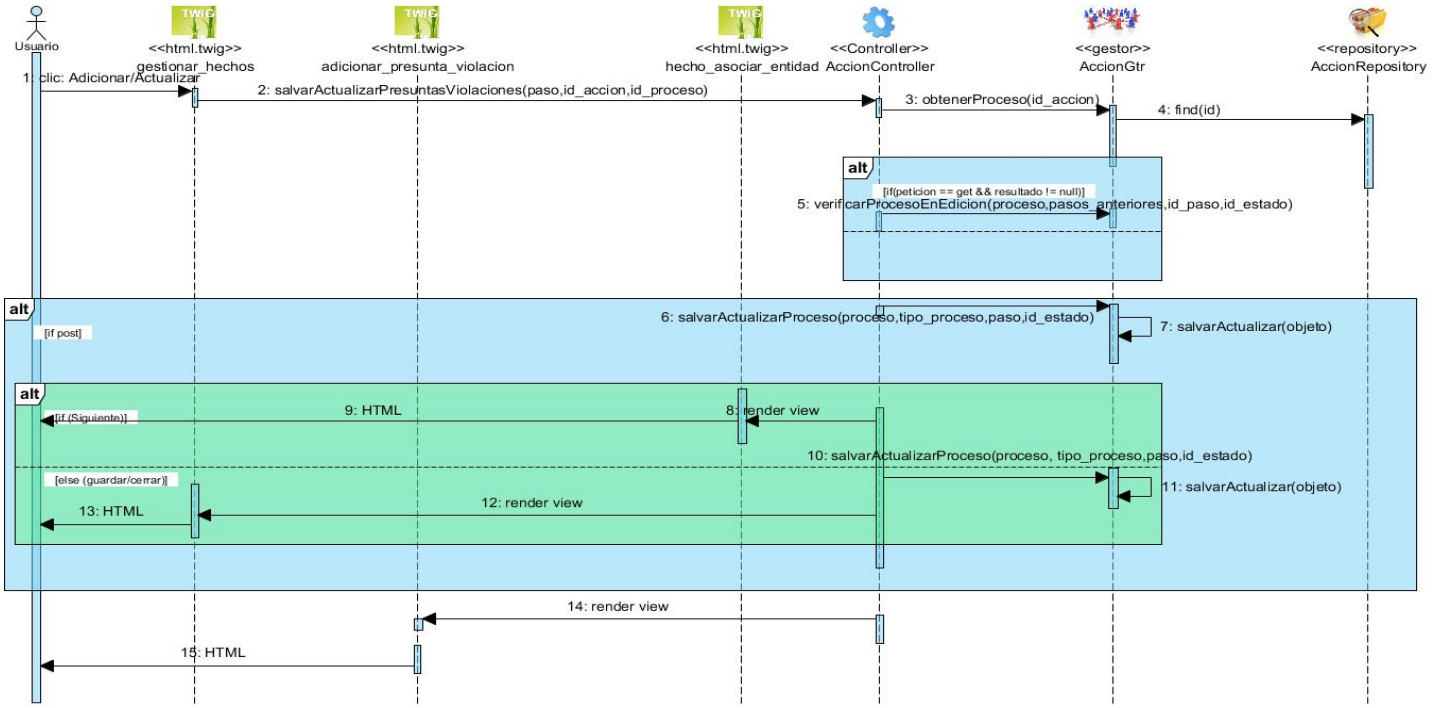
Listar presuntas violaciones



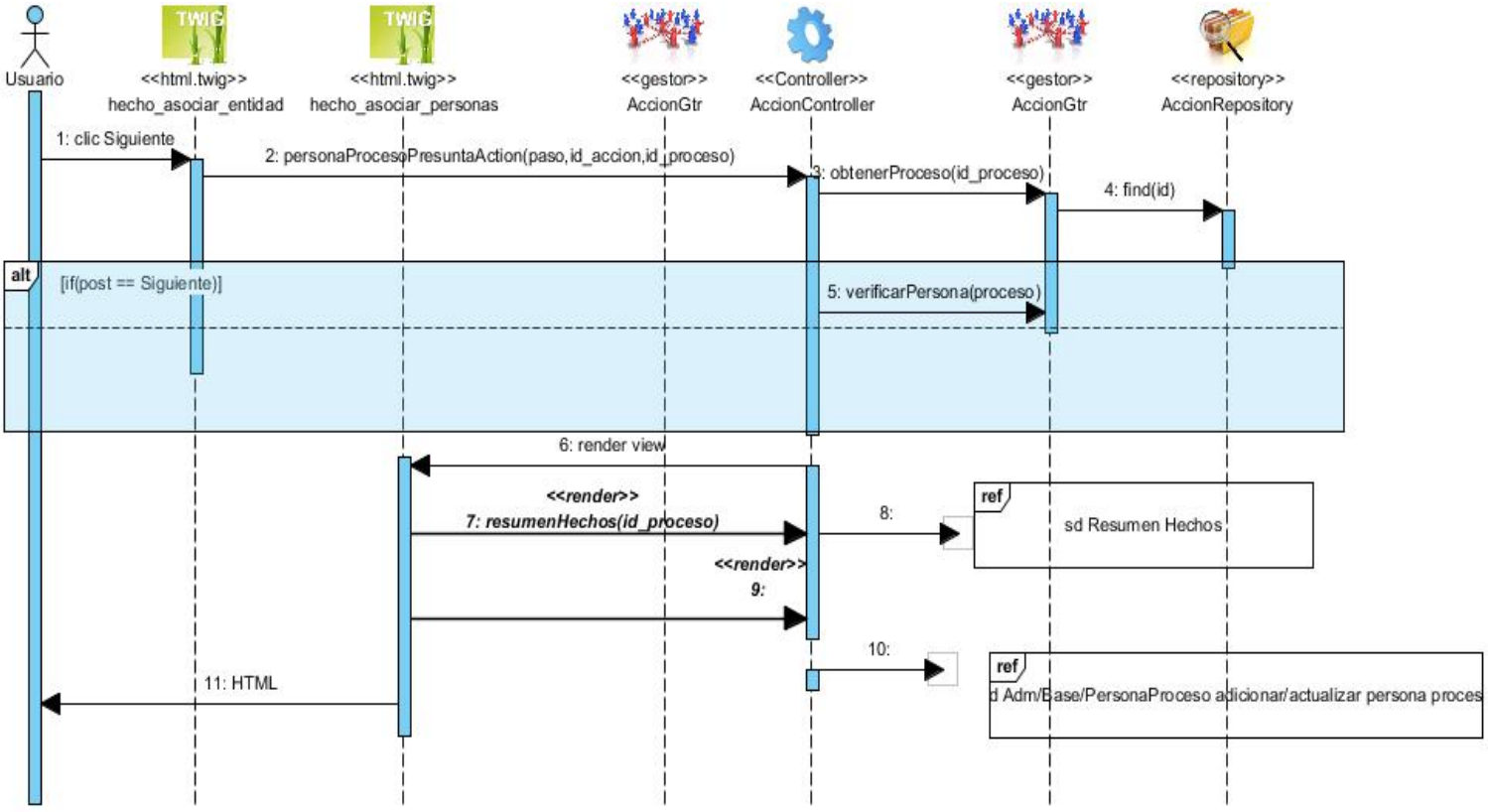
Adicionar/Actualizar presuntas violaciones



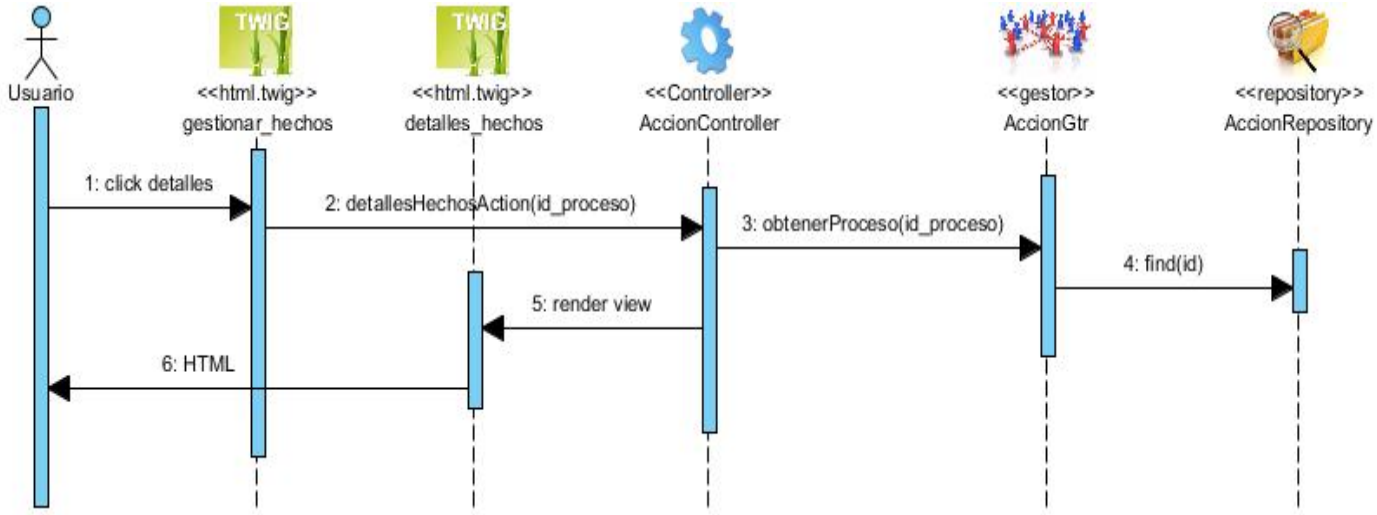
Adicionar/Actualizar Entidad



Adicionar/Actualizar Persona



Detalles de las presuntas violaciones



Anexo # 6: Seguridad para los procesos detección de violaciones y responsables.

La seguridad de los procesos objetos de la investigación se gestionó mediante el archivo security.yml, donde se especificaron las reglas para el control de acceso, los roles y las rutas permitidas para cada usuario. Este archivo es utilizado por todos los módulos de la aplicación.

```
security:
      role_hierarchy:
        ROLE_VF: [ROLE_VF_PLAN_ACCION,ROLE_VF_ESTADO_TAREAS,ROLE_VF_GESTIONAR_DESPACHO,ROLE_VF_NOTIFICAR_DESPACHO,ROLE_VF_RESPUESTA_DESPACHO]
        ROLE_ADMINISTRACION: [ROLE_PROVINCIA_ADM, ROLE_PAIS_ADM, ROLE_MUNICIPIO_ADM]
    firewalls:
        dev:
            pattern:  ^/(_(profiler|wdt)|css|images|js)/
            security: false
        area_segura:
            pattern:    ^/sigef2/
            anonymous: true
            wsse: true
            form_login:
                provider: provider
                check_path: /sigef2/login_check
                login_path: /sigef2/login
                always_use_default_target_path: true
                default_target_path: /sigef2/principal
            logout:
                path:       /sigef2/logout
                target:     /sigef2/login
    access_control:
        - { path: ^/sigef2/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        # ROLES DE ADMINISTRACION
        - { path: ^/sigef2/administracion/dist_geografica/listar_nomencladores, roles: [ROLE_PROVINCIA_ADM, ROLE_MUNICIPIO_ADM, ROLE_PAIS_ADM] }
        # ROLES DE VF
          #Accion
        - { path: ^/sigef2/vf/accion/plan/gestionar_plan, roles: ROLE_VF_PLAN_ACCION }
        - { path: ^/sigef2/vf/accion/plan/estado_tareas, roles: ROLE_VF_ESTADO_TAREAS }
        - { path: ^/sigef2/vf/accion/diligencias/despacho, roles: [ROLE_VF_GESTIONAR_DESPACHO, ROLE_VF_NOTIFICAR_DESPACHO, ROLE_VF_RESPUESTA_DESPACHO] }
```

Anexo # 7: Seguridad para los procesos detección de violaciones y responsables.

También mediante las restricciones en las rutas, con lo que se consigue que los usuarios no puedan acceder a determinadas rutas alterando los parámetros definidos para estas.

```
vf_gestionar_presuntas_violaciones:
    pattern: /presuntas/{paso}/{id_proceso}
    defaults:
        _controller: VFBundle:Accion:gestionarPresuntas
        id_proceso: null
        label: "Acción/Presuntas violaciones de la legalidad"
        padre:
            - 'vf_gestionar_acciones_hechos'

vf_adicionar_presunta_violacion:
    pattern: /presuntas/adicionar_presuntas_violaciones/{paso}/{id_accion}/{id_proceso}
    defaults:
        _controller: VFBundle:Accion:salvarActualizarPresuntasViolaciones
        id_proceso: null
        label: "Acción/Adicionar Presuntas Violaciones de la legalidad"
        padre:
            - 'arq_seg_render_menu_vertical'
    requirements:
        paso: 226
```

Anexo # 8:

Universidad de las Ciencias Informáticas

Minuta de reunión

Autores	Kenia López Hernández Nilson Mulet Morales	Fecha	7/05/2014
Lugar	Laboratorio del Proyecto Sistema de Informatización de la Gestión de las Fiscalías II. Universidad de las Ciencias Informáticas.	Hora Inicio	9:00
Proyecto/ Grupo	Proyecto Sistema de Informatización de la Gestión de las Fiscalías II.	Hora Terminación	11:00

Asunto	Análisis de las variables gestión de la información, control y celeridad.
Asistentes	Teidy Colomar Zaldívar (Fiscal de la Dirección de Verificación Fiscal) Pedro Pablo Cutiño (Jefe de la Dirección de Verificación Fiscal) Kenia López Hernández (Desarrollador del módulo Verificación Fiscal) Nilson Mulet Morales (Desarrollador del módulo Verificación Fiscal)
Ausentes	N/A

Orden del día

1. Análisis del comportamiento actual de las variables control y celeridad en los procesos detección de violaciones y responsables.

Actividades

No	Descripción
1	Se realizó un análisis de la variable control en los procesos detección de violaciones y responsables donde se determinó en conjunto con los fiscales que actualmente no se conocen los detalles del proceso si no se tiene la Resolución de la acción.
2	Se realizó un análisis de la variable celeridad en los procesos detección de violaciones y responsables haciendo uso de un conjunto de indicadores donde se evidenció que para generar documentos el tiempo es de aproximadamente 2 horas.

Partes involucradas en esta entrevista

FGR

Nombre y Apellidos: Teidy Colomar Zaldívar Cargo: Fiscal de la Dirección de Verificación Fiscal

UCI

Nombre y Apellidos: Kenia López Hernández Cargo: Desarrollador del Módulo de Verificación Fiscal	Nombre y Apellidos: Nilson Mulet Morales Cargo: Desarrollador del Módulo de Verificación Fiscal