Universidad de las Ciencias Informáticas

Facultad 3

Título: Diseño e implementación de los procesos de prórrogas, reuniones e informes para el módulo Verificación Fiscal del Sistema de Informatización de la Gestión de la Fiscalía fase II

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

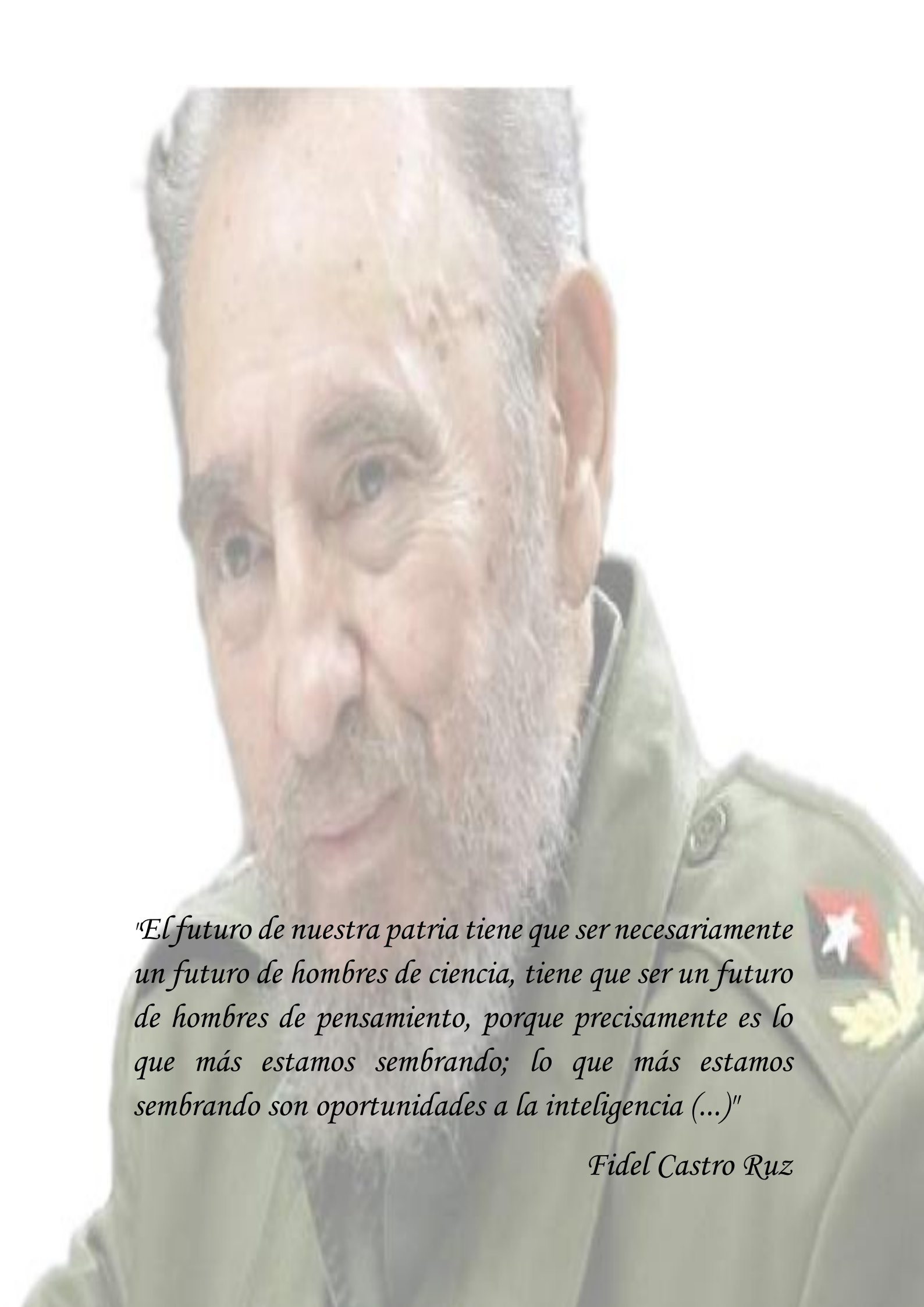
Autor(es): Yoalvy Fernández Martínez
Ledián Elier Lorenzo Ballestero

Tutor(es): Ing. Eduardo Castillo Benítez
Msc. Ana Marys García Rodríguez

Co-tutora: Ing. Claudia Hernández Rizo

La Habana, Junio 2014

"Año 56 de la Revolución"



"El futuro de nuestra patria tiene que ser necesariamente un futuro de hombres de ciencia, tiene que ser un futuro de hombres de pensamiento, porque precisamente es lo que más estamos sembrando; lo que más estamos sembrando son oportunidades a la inteligencia (...)"

Fidel Castro Ruz

Declaración de autoría:

Declaramos que somos los únicos autores del trabajo de diploma “Diseño e implementación de los procesos de prórrogas, reuniones e informes para el módulo Verificación Fiscal del Sistema de Informatización de la Gestión de la Fiscalía fase II” y concedemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste se firma el presente a los _____días del mes de junio del año 2014.

Yoalvy Fernández Martínez

_____

Firma del autor

Ledián Elier Lorenzo Ballestero

_____

Firma del autor

Ing. Eduardo Castillo Benítez

_____

Firma del tutor

Msc. Ana Marys García Rodríguez

_____

Firma de la tutora

Ing. Claudia Hernández Rizo

_____

Firma de la co-tutora

Datos de contacto:

Datos de los tutores:

Nombre: Ing. Eduardo Castillo Benítez

Correo electrónico: ecbenitez@uci.cu

Nombre: Msc. Ana Marys García Rodríguez

Correo electrónico: agarcia@uci.cu

Nombre: Ing. Claudia Hernández Rizo

Correo electrónico: crizo@uci.cu

Datos de los autores:

Nombre: Yoalvy Fernández Martínez

Correo electrónico: yfernandezm@estudiantes.uci.cu

Nombre: Ledián Elier Lorenzo Ballestero

Correo electrónico: leballestero@estudiantes.uci.cu

Agradecimientos:

Yoalvy:

A la Universidad de las Ciencias Informáticas por contribuir a mi formación como profesional.
A los tutores por guiarnos en la investigación y sin cuya ayuda no sería posible la realización de este trabajo.
A mi familia por su apoyo incondicional y la confianza que han depositado en mí.
A todas las buenas amistades que hice durante estos 5 largos años, y son como una familia para mí.
Le agradezco a mi compañero de tesis con el cual pase días y noches trabajando sin descanso para obtener este resultado.
A todos los que me ayudaron de una forma u otra.

Ledián:

Quiero agradecer a las personas que más quiero en la vida, mis padres y mi hermano, por estar siempre pendientes de mí y quererme tanto. A mami por la gran educación que me inculcó desde pequeño, por enseñarme y exigirme ser mejor en cada paso que di. A papi por ser mi mayor ejemplo a seguir y cuidarme tanto. A mi hermano por existir y estar a mi lado siempre, por ser mi mayor tutor en la vida y en la tesis, y darme su ayuda sin importar hora ni lugar. A mi hermana y a mi sobrinita por ser tan especiales para mí. A mi tía y mi primo, por complacerme siempre. A mi novia por saber quererme, entenderme, aguantarme y estar siempre pendiente de mí. A mi padrino por quererme tanto. A mis amigos de la infancia: Osvaldito por apoyarme siempre y hacerme saber que tengo 2 hermanos, a José por ser como es y por todas nuestras andanzas juntos. A mis demás amigos de Camagüey, el Chipri, Raulín, Esildo. A mis amigos del pre, que arrastré conmigo para la UCI, Mastrapa, Riverón y Cocó, por ser 3 hermanitos más. A los amigos hechos aquí que quedarán para siempre, Alejandro, Sanamé, el Yipi, el Puma, Yasel, el Cuadro, y muchos más que sería imposible nombrar, ellos saben quienes son, saben que siempre podrán contar conmigo. Agradezco a Nela, Yanita y Mursu, Yohan, Lester, Adita, Adrián, Osly y a todo el piquete que en este último tiempo me han hecho parte de ellos, por ser amigos y compartir tantos buenos momentos. A Lizandra y familia por hacerme parte de ella. Agradezco también de forma especial a mi compañero de tesis por ser mi amigo, por entendernos y trabajar como equipo. A mis tutores por el apoyo recibido, muchas gracias Ana Marys por aguantarme en tu casa por la madrugada, y prepararme café para desvelarnos y trabajar en el documento. A todos los que se preocuparon por mí y me dijeron en algún momento… ¿y la tesis?

Dedicatoria:

Yoalvy:

A mis padres por su ejemplo, cariño y confianza.
A mi hermano por inspirarme y ser un guía para mí en todos los aspectos de la vida.
A toda mi familia.

Ledián:

A mis padres y hermano por serlo todo para mí, sé que este logro mío es un orgullo para ellos.

Resumen

La Fiscalía General de la República de Cuba constituye el órgano del Estado encargado de velar por la observancia de la legalidad, así como de combatir las manifestaciones de corrupción en los Organismos del Estado y sus dependencias. La misma cuenta con el área Verificación Fiscal donde se ejecutan los procesos de prórrogas, reuniones e informes, los cuales se ven afectados respecto al tiempo de ejecución y la falta de control. En aras de solucionar dichos problemas, se identifica la necesidad de implementar un sistema informático que contribuya a mejorar la celeridad y el control de los procesos antes mencionados.

Como fundamento de la investigación se realiza un análisis en torno a las actividades que se desarrollan en los procesos de prórrogas, reuniones e informes, el estado del arte de los sistemas que los implementan y las tecnologías para el desarrollo de la solución propuesta. A raíz del análisis de la arquitectura a emplear, se obtiene como resultado el diseño de los requisitos del sistema. Además, a partir del entorno de desarrollo concebido, se realiza la implementación de los componentes diseñados y se ejecutan las pruebas de software para garantizar la calidad del producto.

La solución obtenida pone a disposición de los fiscales un sistema que permite ejecutar los procesos de prórrogas, reuniones e informes de Verificación Fiscal disminuyendo el tiempo de ejecución de los mismos y garantizando el control sobre la información que se gestiona.

Palabras claves: celeridad, control, informes, prórrogas, reuniones, verificación fiscal.

Tabla de Contenido

Introducción

El acelerado avance de las Tecnologías de la Información y las Comunicaciones (TIC), ha traído consigo múltiples cambios en la economía, la sociedad y la política, contribuyendo al buen desarrollo y avance de estas esferas. Las TIC agrupan los elementos y las técnicas usadas en el tratamiento y la transmisión de las informaciones, principalmente en las ramas de informática, internet y telecomunicaciones. (1)

Son varios los países y gobiernos que han optado por la implantación de sistemas informatizados para el control de sus procesos internos. Las TIC ayudan a mejorar los sistemas institucionales existentes de un gobierno de modo que puedan reestructurarse, dando pie a nuevos planes innovadores que allanen el camino para un gobierno colaborador, eficaz, transparente y responsable, fundamental para un desarrollo sostenible. (2)

Cuba, a pesar de ser un país subdesarrollado, ha dado pasos gigantescos en el desarrollo de las TIC, por lo que la Fiscalía General de la República (FGR) en virtud de aprovechar el vertiginoso desarrollo identificó la necesidad de informatizar sus principales procesos, lo cual permitirá un mayor control en las informaciones obtenidas y generadas, facilidades en su almacenamiento, optimización del recurso tiempo, así como otras ventajas inherentes al proceso de informatización las cuales se revierten en una mejor gestión interna del trabajo a realizar. Como resultado del proceso de informatización es desarrollado en la Universidad de las Ciencias Informáticas (UCI), específicamente en el Centro de Gobierno Electrónico (CEGEL), el Sistema de Informatización de la Gestión de la Fiscalía (SIGEF) dirigido a informatizar los diversos procesos desarrollados en cada una de las instancias de la fiscalía.

"La FGR 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)

La FGR consta de varias áreas, una de ellas es Verificación Fiscal (VF), donde se desarrollan diversos procesos entre los se encuentran: prórrogas, reuniones e informes. En dicha área se planifican y ejecutan verificaciones e investigaciones fiscales.
Una verificación fiscal es la comprobación que se le realiza a una determinada entidad para evaluar el cumplimiento de las leyes y disposiciones legales en la misma. Una investigación persigue los mismos objetivos que una verificación, su diferencia

fundamental radica en que las investigaciones disponen de un tiempo menor para su desarrollo. (4)

Durante la ejecución de los procesos de prórrogas, reuniones e informes de las verificaciones e investigaciones fiscales se genera un alto volumen de información, lo que conlleva a que al finalizar cada acción los expedientes excedan generalmente las 100 páginas y estos documentos se desarrollan manualmente por los fiscales. Además el almacenamiento de toda esta información se complejiza porque al encontrarse en formato duro (papel) ocupa un espacio considerable y con el paso del tiempo se va deteriorando, por lo que los documentos que permanecen más de 5 años en la fiscalía y pierden su relevancia, son destruidos. El envío de expedientes, documentación y notificaciones a cada instancia en caso de ser necesario, se realiza por correo electrónico o tradicional, que requiere de más tiempo si el mismo es devuelto para que se realicen otras investigaciones.

El sistema de trabajo antes descrito provoca:

- Necesidad de tiempo y esfuerzo considerable por parte de los fiscales para la creación de documentos.
- Dilación en la ejecución del flujo de los procesos en las aprobaciones y disposiciones entre instancias, así como en la generación y análisis de reportes.
- Deterioro y pérdida de los expedientes al paso del tiempo debido a su almacenamiento en formato duro únicamente, dificultando su conservación por períodos prolongados de tiempo.
- Consultas engorrosas a los documentos o expedientes archivados, provocando un acceso y análisis inextricable a los registros de antecedentes, para la ejecución de procesos actuales.
- Dificultad para supervisar y controlar en tiempo real los procesos fiscales y el trabajo de las instancias provinciales y municipales en cuanto a la obtención de reportes estadísticos sobre sus procesos.
- Verificación constante del tiempo que resta para culminar cada caso, debido a la existencia de términos que no pueden ser violados y a la carencia de sistemas de alertas que recuerden las fechas límites.

En concordancia con la situación problemática expuesta anteriormente se identifica el siguiente **problema a resolver:** ¿cómo gestionar la información de los procesos de prórrogas, reuniones e informes en el área de Verificación Fiscal de manera que contribuya a un mayor control y celeridad en su ejecución?

Definiéndose como **objeto de estudio:** el proceso de desarrollo de software de gestión en la Informática Jurídica, enmarcándose en el **campo de acción:** proceso de desarrollo de software de gestión en los procesos fiscales.

En función de dar respuesta al problema identificado se traza como **objetivo general:** diseñar e implementar los procesos de prórrogas, reuniones e informes en el área de Verificación Fiscal, de manera que permita un mayor control y celeridad en su ejecución.

El objetivo general se desglosa en los siguientes **objetivos específicos:**

1. Elaborar el marco teórico y el estado del arte de la investigación para identificar tendencias y adoptar una posición al respecto.
2. Diseñar los procesos de prórrogas, reuniones e informes del módulo Verificación Fiscal del proyecto SIGEF fase II, para transformar los requisitos en un diseño adecuado para la construcción del sistema.
3. Implementar los procesos de prórrogas, reuniones e informes del módulo Verificación Fiscal del proyecto SIGEF fase II para la obtención del producto.
4. Validar la solución propuesta para evaluar el cumplimiento del objetivo de la investigación.

Para dar cumplimiento a los objetivos planteados se definieron las siguientes **tareas de la investigación**:

1. Análisis del proceso de desarrollo de software, utilizado en el proyecto SIGEF fase II, específicamente cómo se realizan las disciplinas de diseño, implementación y pruebas.
2. Análisis de los procesos que se realizan en las prórrogas, reuniones e informes en la verificación e investigación fiscal.
3. Estudio del estado del arte a sistemas informáticos que aborden procesos de Verificación Fiscal.
4. Análisis de los artefactos generados como parte del modelado del negocio para comprender el funcionamiento de la organización.
5. Análisis de la especificación de requisitos y los prototipos funcionales que modelan el funcionamiento de la organización.

6. Diseño del diagrama de clases persistentes, de vistas, de controladoras y gestoras, de secuencias y de repositorio de los procesos de prórrogas, reuniones e informes del módulo Verificación Fiscal.
7. Validación del diseño mediante la aplicación de métricas de diseño.
8. Implementación de las funcionalidades de los procesos de prórrogas, reuniones e informes del módulo Verificación Fiscal.
9. Validación de la implementación del sistema a través de pruebas de caja blanca y caja negra, para verificar su correcto funcionamiento.
10. Obtención de datos reales de la ejecución manual de los procesos de prórrogas, reuniones e informes, mediante una entrevista a fiscales.
11. Validación de las variables de la investigación frente al problema planteado.

Se plantea como **idea a defender**: el diseño e implementación de los procesos de prórrogas, reuniones e informes en el área de Verificación Fiscal, contribuirá a un mayor control y celeridad en su ejecución.

Métodos teóricos:

Observación: Con este método fue posible recopilar una serie de datos confiables que ayudan a la comprensión del fenómeno y a la definición de lo primordial de la problemática. Al mismo tiempo esto posibilitó el planteamiento del problema permitiendo enmarcar el objeto de estudio y el campo de acción, lo cual propicia enfocar la investigación hacia lo que se necesita alcanzar y cómo alcanzarlo.

Analítico-sintético: permitió integrar los conocimientos del proceso de desarrollo de software para su aplicación en la informática jurídica, principalmente en la FGR, además de realizar un procesamiento de la información y llegar a conclusiones prácticas y teóricas sobre la investigación.

Modelación: posibilitó la creación de los principales artefactos de cada una de las fases incluidas en el desarrollo de la solución, así como la implementación de los modelos de diseño e implementación.

Estructura del documento:

Capítulo 1. Fundamentación teórica: muestra un análisis de algunos de los sistemas de gestión fiscal desarrollados en la actualidad, además expone las diferentes

herramientas, lenguajes y metodologías utilizadas para dar solución al problema planteado.

Capítulo 2. Requisitos y diseño: se describen los requisitos funcionales y no funcionales que abarcan los procesos en cuestión. Refleja el diseño realizado a través de los diagramas de clases persistentes, de las vistas, de las clases controladoras, los diagramas de secuencia; haciendo uso de patrones de diseño, así como su validación a través de las métricas propuestas.

Capítulo 3. Implementación y prueba: abarca lo referente al modelo de implementación del sistema en términos de componentes, dentro de los que se incluyen ficheros de código fuente, scripts y ejecutables. Además se reflejan los resultados de la validación de la implementación a través de pruebas de caja negra y caja blanca que garantizan que el sistema satisface las necesidades del cliente.

Capítulo 1. Fundamentación teórica

1.1. Introducción

En el presente capítulo se abordan todos los elementos desde el punto de vista teórico que se tienen en cuenta para el desarrollo de la solución propuesta en las fases de diseño, implementación y prueba. Quedan reflejados los conceptos fundamentales para un mejor entendimiento del negocio dentro de los procesos de prórrogas, reuniones e informes en el área de VF. Se abordan las principales características de varias aplicaciones jurídicas desarrolladas en el mundo que sirvieron de punto de partida para realizar un análisis previo y obtener elementos útiles para el desarrollo del sistema requerido. Se identifican y describen los lenguajes y herramientas utilizadas en el presente trabajo, así como los patrones, métricas de diseño, y las pruebas a realizar.

1.2. Conceptos fundamentales

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, entre los cuales se encuentra el de Protección de los Derechos Ciudadanos. (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. Las Fiscalías Municipales en que las necesidades del servicio así lo determinen, podrán organizarse en secciones, que se integran con varios fiscales, para la atención de esferas de trabajo especializadas. (3)

Proceso judicial: conjunto de actuaciones que realiza un tribunal de justicia en un procedimiento judicial desde su inicio hasta que se dicta sentencia. (3)

Verificación Fiscal: la verificación fiscal constituye el método principal que emplea la fiscalía para comprobar el cumplimiento de la Constitución, las leyes y demás disposiciones legales en las instituciones y sus entidades independientes, formulando los pronunciamientos que resulten procedentes para que se restablezca la legalidad quebrantada, accionar contra los infractores, actuando como medio de educación jurídica y de prevención de nuevas infracciones. La verificación fiscal se sustancia

mediante las distintas acciones y diligencias que realizan los fiscales, con el auxilio de especialistas y peritos en la materia objeto de inspección, organizándose en orden cronológico en el expediente. (4)

Expediente de Verificación Fiscal: conjunto de datos o conjunto de incidencias recopilados por el fiscal durante el trámite de un proceso de verificación fiscal. (3)

1.3. Descripción actual de los Procesos.

Las verificaciones e investigaciones fiscales son los instrumentos principales que posee la FGR para el cumplimiento del mandato constitucional, de velar por la observancia de la legalidad y combatir las manifestaciones de corrupción en los Organismos del Estado y sus dependencias, las direcciones subordinadas a los órganos locales del Poder Popular, las demás entidades económicas y sociales y por los ciudadanos, formulando en los casos necesarios los pronunciamientos que resulten procedentes. (5)

La FGR posee una dirección de VF que tiene a su cargo la dirección y el control de la realización de las verificaciones e investigaciones fiscales, por los órganos de la Fiscalía en los organismos del Estado, sus dependencias y direcciones subordinadas a los órganos locales del Poder Popular y demás entidades económicas y sociales.
Las verificaciones e investigaciones fiscales reciben la clasificación de acción, dentro de la cual se procesan disímiles documentos (expedientes) y se ejecutan varios procesos como son los de prórrogas, reuniones e informes. (6)

El fiscal del departamento municipal de VF (FMVF) es el encargado de llevar a cabo el procedimiento pertinente a la verificación fiscal, detectando ya sean hechos delictivos, indicios de enriquecimiento indebido, entre otras violaciones; a partir de la radicación del expediente de verificación fiscal donde se almacena todo lo referente al proceso, se practican las diligencias necesarias para demostrar las violaciones de la legalidad y las causas y condiciones que la generaron. En caso de necesitar un mayor tiempo del establecido para continuar con la tramitación del expediente puede solicitar una prórroga de hasta 30 días mediante un escrito dirigido al jefe del Departamento de VF, creando la Solicitud de Prórroga. La solicitud es valorada por el fiscal jefe Municipal (FJM), quien valora si procede o no, en caso de ser denegada la solitud se notifica de la decisión al FMVF, que da por terminado el proceso, en caso contrario la solicitud es elevada a través de correo electrónico o postal al Departamento Provincial de VF para su

aprobación. Luego de ser analizada la solicitud, se notifica a través del Escrito (Autorizando/Denegando) Prórroga a la Tramitación de la VF a la instancia inferior. (6)

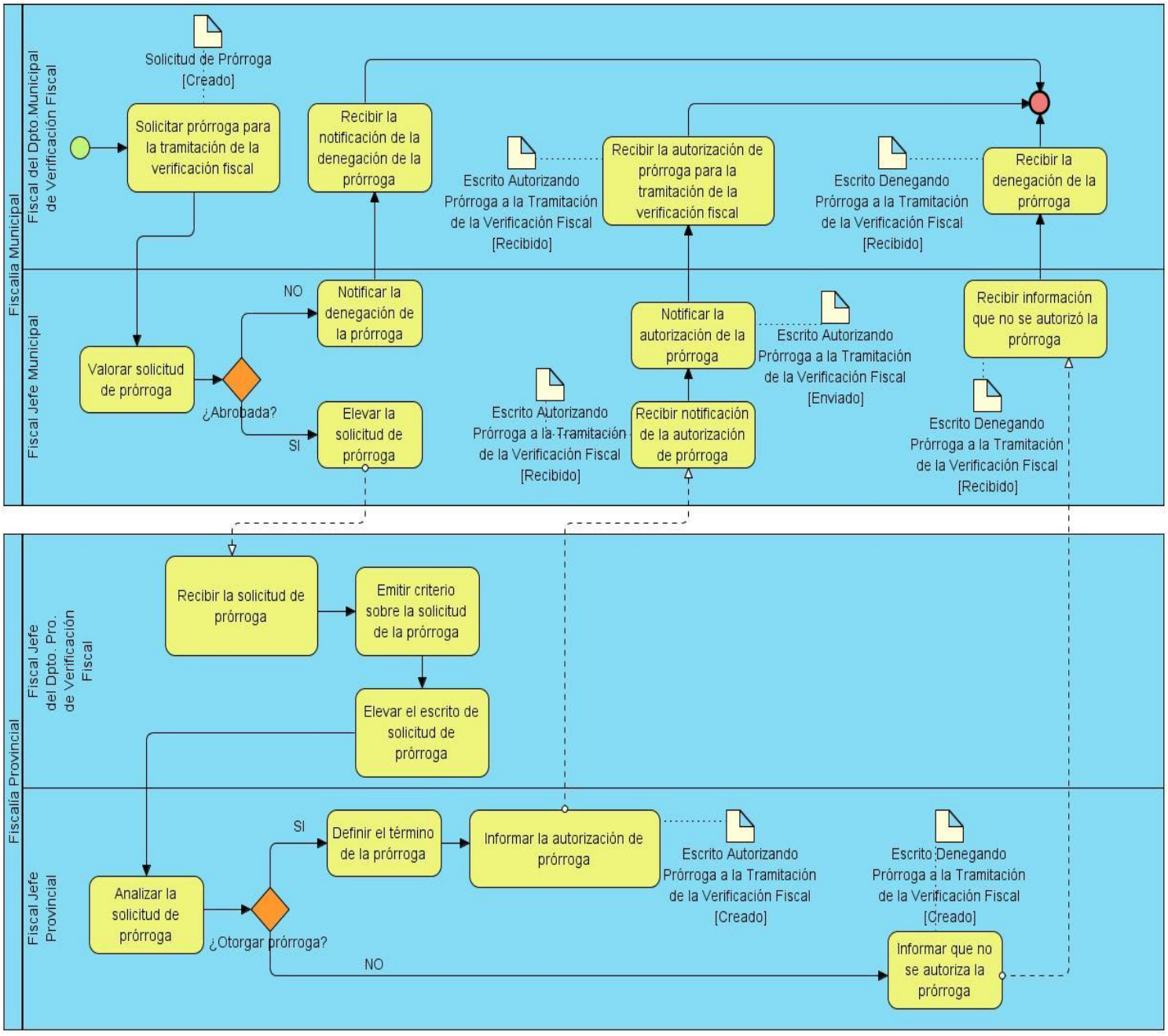


Figura 1. Subproceso Prórroga a la Tramitación del Expediente de Verificación Fiscal

Fuente: Expediente Proyecto SIGEF II

Una vez concluida la verificación se genera un informe por parte de los especialistas con los resultados del proceso, el FMVF hace una valoración del contenido del informe y emite su aprobación, la cual da paso a la Reunión Previa y se genera el Acta de Reunión Previa de la verificación y el Informe Resumen de los Resultados de la misma. Después se realiza la Reunión de Conclusiones, si no existen discrepancias o son erradicadas en la misma reunión, se confecciona el Acta de Reunión de Conclusiones, y se da a conocer los resultados a los trabajadores, en caso de no darle solución a las discrepancias en la Reunión de Conclusiones se realiza con un plazo de 5 días una re-comprobación a la entidad, o puede solicitar una prórroga para re-comprobar las discrepancias surgidas siguiendo el proceso anteriormente explicado. Luego de ser concluido estos subprocesos el FMVF realiza una Resolución de la VF dejando constancias de todo el proceso. (6)

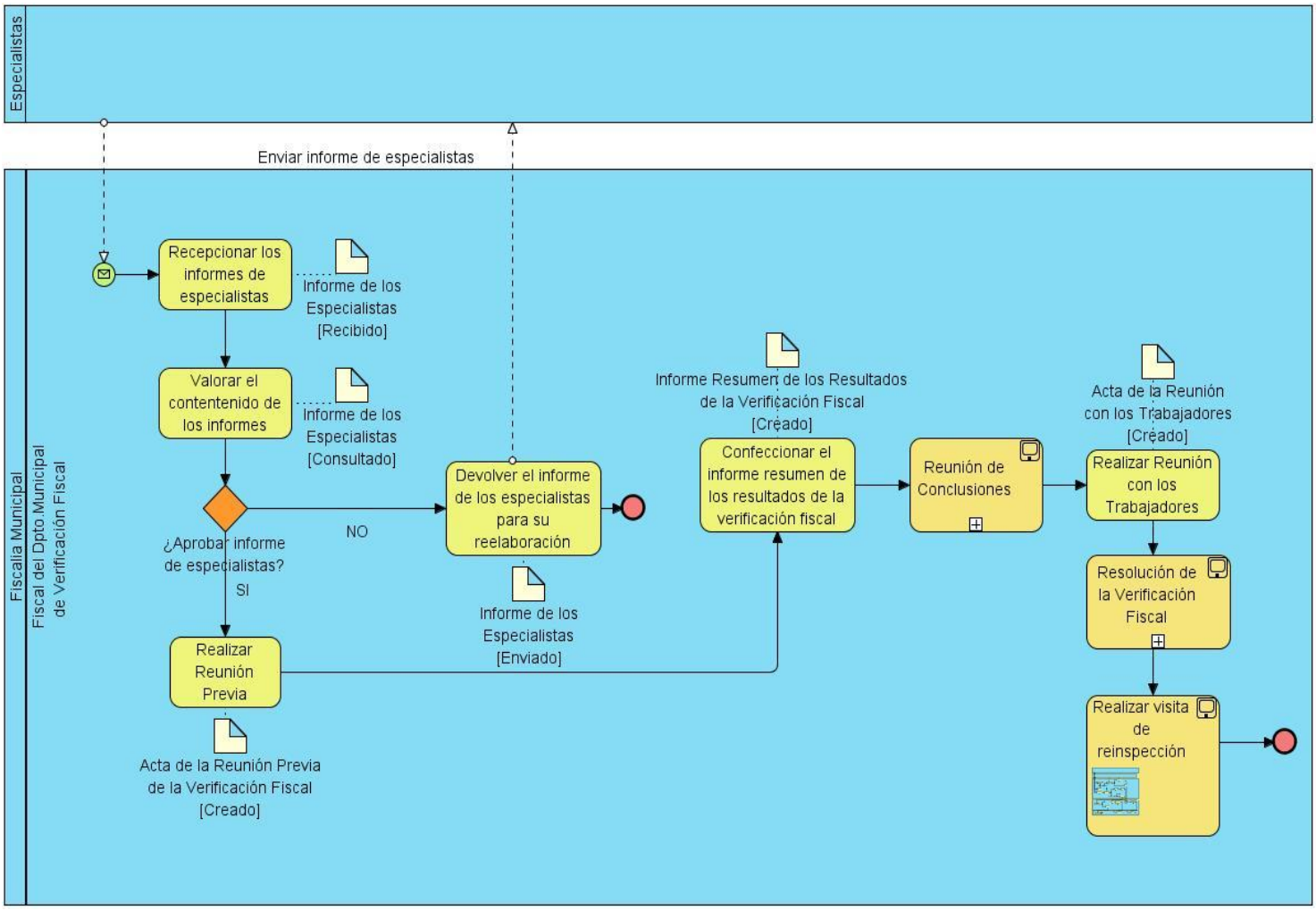


Figura 2. Proceso Conclusión de la Verificación Fiscal.

Fuente: Expediente Proyecto SIGEF II

1.4. Valoración del estado del arte de sistemas de gestión fiscal

En las fiscalías a nivel mundial se llevan a cabo engorrosos procesos, por lo que a través del uso eficiente de las tecnologías de la información, en muchos de estos países se ha logrado crear sistemas fiscales informatizados para lograr un mayor control y celeridad del almacenamiento y procesamiento de la información, así como adquirir una mayor transparencia en sus políticas fiscales.
La FGR cuenta actualmente con un sistema denominado Sistema de Informatización de la Gestión de la Fiscalía Fase I, pero no posee un módulo para el área VF, por lo que se hace necesario realizar un estudio de los sistemas de informática jurídica existentes en el mundo con el objetivo de analizar si estos softwares proporcionan una solución total o parcial al problema en cuestión para así identificar tendencias y adoptar una posición al respecto:

- **SAF:**

Sistema de Apoyo a los Fiscales (SAF), es un sistema computacional que apoya la labor de investigación y posterior desarrollo del proceso penal, a través de las distintas opciones y funcionalidades contenidas en los módulos que lo componen. Permite administrar un gran número de causas penales, con agilidad y transparencia. Realiza un manejo eficiente de grandes volúmenes de información y trabajo colaborativo entre los equipos involucrados. Presenta un módulo (Módulo de Gestión) para apoyar al Fiscal en la ejecución y control de las actividades que deba realizar dentro del curso de una investigación, permitiendo el registro de información (diligencias, solicitudes, decisiones, constancias, citaciones) que se estime relevante para el desarrollo del proceso penal. El sistema es usado en varios países latinoamericanos fundamentalmente en Chile. (7)

El SAF no es una aplicación idónea a implementar en Cuba, ya que a pesar de poseer funcionalidades para gestionar información e investigaciones, estas no cuentan con la gestión de actividades que son llevadas a cabo según las leyes cubanas, como son los procesos de prórroga a la tramitación de un expediente y las reuniones.

- **Fortuny:**

El sistema ha sido desarrollado por el Ministerio de Justicia y la Fiscalía General de España para facilitar la realización de los procedimientos judiciales. El sistema Fortuny da soporte mediante flujos de tramitación a las diferentes tareas de gestión documental

y gestión de trabajos necesarios para los diferentes órdenes judiciales. La aplicación recoge todo el proceso completo desde el registro del asunto en el inicio de la tramitación, la generación, gestión y almacenamiento de todos los documentos producidos, así como la evolución de toda la información controlada en todo momento por la definición que se haya realizado, asegurando la coherencia en la tramitación de la fiscalía. También incorpora información sobre trámites procesales, informes, escritos de acusación, señalamientos y sentencias. Esta aplicación moderna y funcional se utiliza actualmente en la mayor parte de las fiscalías de España. (8) (9)

Dicho sistema permite gestionar información de sistemas fiscales, pero con características particulares que difieren de las necesidades de la FGR pues no maneja procesos como verificaciones fiscales.

- **Lex-Doctor:**

Lex-Doctor es un sistema integral de gestión jurídica desarrollado por Sistemas Jurídicos SRL[1], empresa que tiene un importante posicionamiento en el segmento de los sistemas aplicados a la actividad jurídica en Latinoamérica. Permite consultar expedientes en línea y en tiempo real y gestionar cada uno en una única base de datos centralizada. Además posee herramientas de búsqueda, clasificación, y análisis de datos con listados, reportes e informes personalizados, abarcando la gestión de causas, con tecnología de última generación, escalable y adaptable a las necesidades de cada organismo. (10)

Lex-Doctor a pesar de ser un sistema con prestigio y aceptación, presenta inconvenientes para su utilización por parte de la FGR pues a pesar de ser un software escalable y centrar su trabajo sobre la gestión documental fundamentalmente, no presenta funcionalidades para la gestión de verificaciones e investigaciones.

- **KIWI:**

El Ministerio Público Fiscal en Panamá ha desarrollado y puesto en funcionamiento el sistema informático KIWI. Dicho software permite gestionar digitalmente los casos y solicitudes administrativas que se diligencian internamente en la institución. Abarca desde la toma de denuncias por distintas vías hasta el análisis jurídico, diligenciamiento, investigaciones y judicialización de los diferentes casos jurisdiccionales. (11)

El software abarca procedimientos pertinentes a investigaciones, pero no dispone de las funcionalidades necesarias para realizar una gestión de información completa como se

[1] SRL: Sociedad de Responsabilidad Limitada.

refleja en las necesidades de la FGR, además de que estas no abarcan procesos de prórrogas, ni reuniones.

Después de realizar un estudio de los sistemas informáticos para la fiscalía anteriormente enunciados, se puede afirmar que son sistemas de gran éxito y uso, que responden a las necesidades del sistema judicial para los que fueron creados, y aunque se pueden tomar algunas ideas, estos no satisfacen las necesidades de la FGR, pues no son lo suficientemente flexibles para adaptarlos a otras leyes. Los creadores de los mismos mantienen los derechos sobre sus productos, y pueden acceder a información sensible para el país. Además se trata en todos los casos de sistemas privativos, cuestión esta que entra en contradicción con las políticas de migración a software libre que se está realizando en Cuba, particularmente en la FGR, la cual tiene como base la soberanía tecnológica. Son sistemas personalizados que no se ajustan al negocio y a las necesidades de las fiscalías cubanas porque a pesar de que permiten la gestión de información no se realizan procesos específicos como las prórrogas y reuniones necesarios en la ejecución de una verificación o investigación fiscal, ni generan alertas a los diferentes niveles de la fiscalía.

1.5. Características del Proyecto SIGEF II

El Sistema de Informatización de la Gestión de las Fiscalías II, está dirigido a adicionar nuevas funcionalidades al SIGEF fase 1 anteriormente implementado, lo que permitirá aumentar el nivel de informatización de los procesos fiscales, condicionando un incremento en la calidad de la tramitación, supervisión y control en tiempo real de los mismos, una reducción de los términos de las actividades y diligencias practicadas, de la utilización de documentos en formato duro, un mayor control de la observancia de la garantía de los procesados, utilización óptima de la fuerza fiscal, posibilitando mejorar la calidad, rapidez y profesionalidad de los servicios, que brinda la FGR en sus diferentes instancias, así como incrementar los niveles de gestión de la información, garantizar su adecuada organización y preservación facilitando la toma de decisiones a los diferentes niveles del Estado Cubano. (5)

La conformación de la plataforma para el desarrollo es Linux Apache Postgres PHP, la cual se selecciona tomando como punto de partida los requisitos no funcionales, de manera que esté en correspondencia con la política de migración hacia software libre que tiene la FGR, enmarcada en la estrategia del país que tiene su base en la soberanía tecnológica. (12)

La plataforma seleccionada tiene gran prestigio internacional, teniendo una base sólida en el desarrollo de las aplicaciones web actuales. Para su implementación se utilizaron marcos de trabajo que garantizan una solución más completa con la incorporación de buenas prácticas y patrones de diseño sobre la base de los servicios y estructuras que se implementan en el subsistema de Arquitectura Base, donde se define la utilización del patrón Modelo Vista Controlador para el desarrollo de la aplicación en cada módulo. (12)

La arquitectura utilizada en el proyecto tiene como objetivo estandarizar y agilizar la construcción del sistema. Esta sustenta su base sobre una serie de componentes orientados en dos perspectivas: horizontal y vertical, las cuales están definidas para cumplir con los requisitos funcionales en módulos y algunos de los requisitos no funcionales en niveles. Esta arquitectura agiliza el desarrollo haciendo uso de patrones, permitiendo el desarrollo del trabajo en paralelo. (12)

La arquitectura está basada en el patrón arquitectónico Modelo Vista Controlador donde cada módulo hace uso de las tres capas que propone el patrón y se le adiciona además una capa denominada negocio, que es la encargada de encapsular la lógica del negocio. (12)

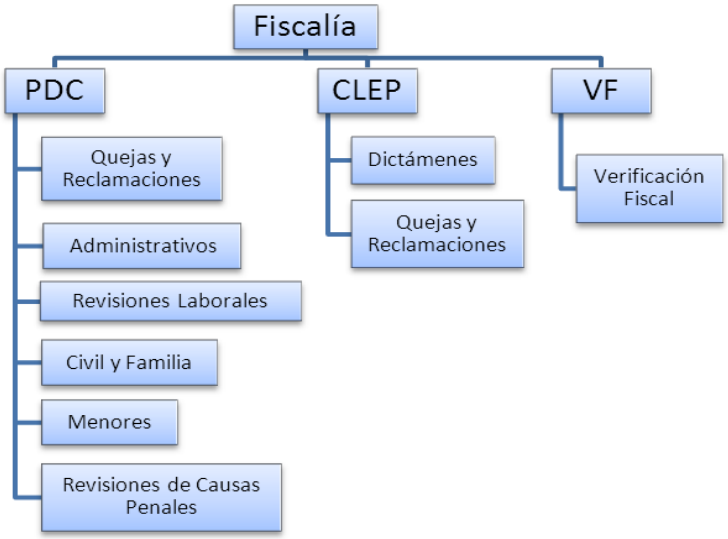


Figura 3: Representación Horizontal de la Arquitectura.

Fuente: Expediente Proyecto SIGEF II

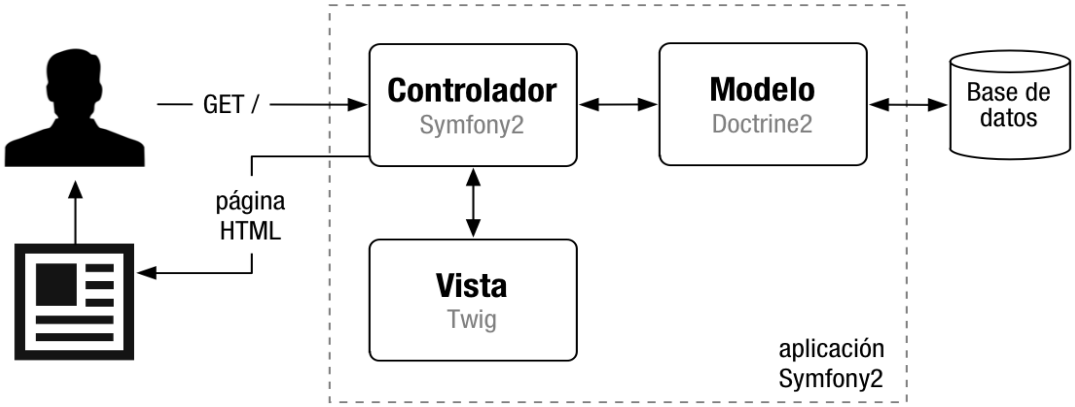


Figura 4: Representación vertical de la arquitectura de cada módulo
Fuente: Expediente Proyecto SIGEF II

El subsistema de arquitectura base recoge el diseño y la implementación de todas las funcionalidades comunes en los diversos módulos. Existen componentes para la creación de objetos entre niveles y módulos, para asegurar el manejo transaccional adecuado según el tipo de sistema que se implementa, para el control de excepciones y trazas, mensajería, componentes para funcionalidades visuales que se identifican como recurrentes, para asegurar la integralidad, seguridad y el enrutamiento. De esta manera no solo se reduce la implementación a los requisitos funcionales básicos, sino que el código es altamente reutilizable y se limitan los errores o vulnerabilidades en el mismo. (12)

El 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. (12)

1.5.1. Modelo de desarrollo de software

En el contexto de la ingeniería de software para dar solución de forma óptima a determinado problema es necesario trazar una estrategia de desarrollo que acompañe al proceso, métodos y tenga herramientas. Los estándares establecen los diferentes

procesos implicados a la hora de desarrollar y mantener un sistema desde que surge la idea o necesidad de desarrollar las aplicaciones hasta que éstas se retiran de explotación. Sin embargo, ninguno impone un modelo de procesos concreto (modelo de ciclo de vida), ni cómo realizar las diferentes actividades incluidas en cada proceso ya que estas están en estrecha concordancia con las necesidades y características muy únicas de cada proyecto; por lo que a nivel mundial existen empresas que no usan modelos, ni metodologías, otras que usan metodologías propias y otras que utilizan agrupaciones de modelos.

Proceso Unificado de Desarrollo (RUP)

RUP es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto. (13)

El ciclo de vida de vida de RUP se resume en tres características fundamentales:

- Dirigido a casos de uso.
- Centrado en la arquitectura.
- Iterativo e incremental.

Esta metodología integra el desarrollo de un sistema en cuatro fases y nueve flujos de trabajo, de los cuales seis se conocen como flujos de ingeniería y los restantes como disciplinas de apoyo. (13)

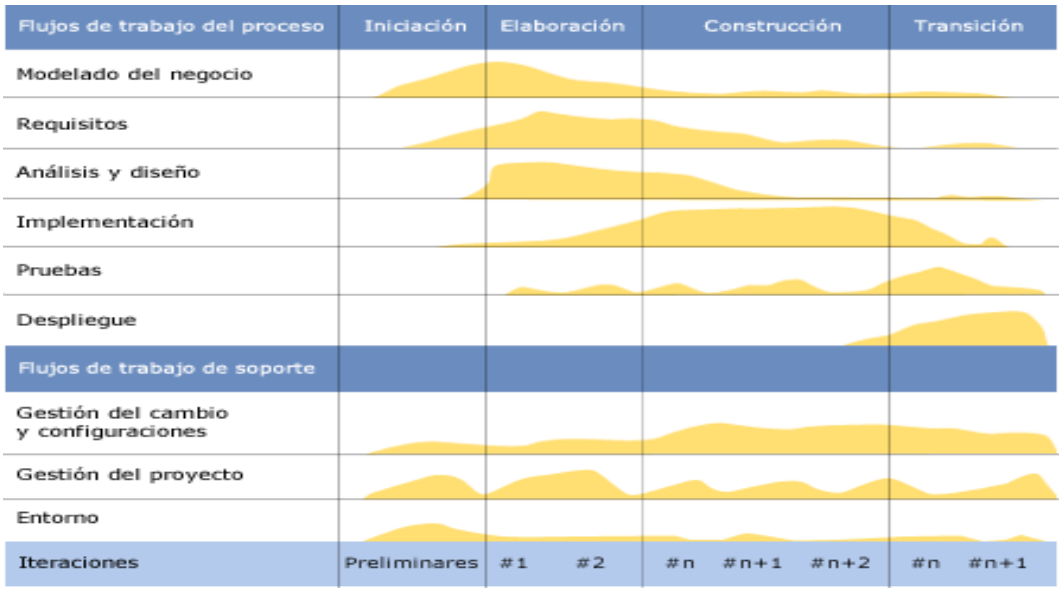


Figura 5: Diagrama de incidencias de las fases y las disciplinas propuestas por RUP

Para el desarrollo de SIGEF II se adoptó la metodología de desarrollo RUP pues se definió una guía que permitió establecer las disciplinas del desarrollo del software, las actividades que se realizaron, los artefactos que se generaron en cada una de estas, así como sus responsables. Además de gestionar y organizar los recursos vinculados al proceso de producción.

Como parte de una estrategia adoptada por los especialistas del proyecto, se decide omitir una de las características de RUP: “Dirigido a casos de uso”, y modelar por procesos, lo cual posibilita el ahorro considerable de tiempo, sin afectar el desarrollo del producto. Además se incorporan los distintos subprocesos dictados por el nivel 2 de Integración de modelos de madurez de capacidades (CMMI) establecidos en el programa de mejora propuesto por la UCI. Esta certificación fue obtenida en 2011 por CEIGE (Centro de Informatización de la Gestión de Entidades), reconocida por el SEI (*Software Engineering Institute*) como aval de calidad de sus procesos de software. EL uso del programa de mejora permite manejar eficientemente proyectos a largo plazo, haciendo énfasis en la generación de artefactos bien documentados, posibilitando la capacitación y transferencia del producto. (14)

El desarrollo del presente trabajo se centrará en las fases de diseño, implementación y pruebas, con la elaboración de los artefactos pertinentes para obtener el producto o la aplicación final. Estos artefactos a generar son el modelo de diseño, los diagramas de clases persistentes, gestoras, controladoras, vistas, así como los diagramas de secuencias y los ficheros de código.

1.6. Descripción de lenguajes y herramientas a utilizar

1.6.1. Lenguajes

Lenguaje unificado de modelado (UML) 8.0

El Lenguaje de Modelado Unificado (UML - *Unified Modeling Language*) es un lenguaje que permite modelar, construir y documentar los elementos que forman un producto de software que responde a un enfoque orientado a objetos. Es un estándar internacional para definir, organizar y visualizar los elementos que configuran la arquitectura de una aplicación orientada a objetos. UML es un lenguaje de propósito general de modelado visual que permite una abstracción del sistema y sus componentes, además de unificar las experiencias acumuladas sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. (15)

Se emplea esta notación ya que proporciona un diseño sólido, responde a un enfoque orientado a objetos y es soportado por la herramienta seleccionada para el diseño.

PHP 5.3

PHP (*Hypertext Preprocessor*) es un lenguaje de código abierto del lado del servidor muy popular especialmente adecuado para el desarrollo web de contenido dinámico. PHP puede ser desplegado en casi todos los sistemas operativos y plataformas sin costo alguno, soporta la mayoría de los servidores web existentes destacando que es el módulo Apache más popular entre las aplicaciones que utilizan Apache como servidor web y además ofrece soporte para diversos gestores de base de datos entre los que se encuentra PostgreSQL. (16)

La programación en PHP es segura y confiable ya que el código fuente escrito en PHP es invisible al navegador y al cliente pues es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Permite a los desarrolladores aplicar cualquier técnica de desarrollo que le permita escribir código ordenado, estructurado y manejable, como lo es el patrón de diseño modelo-vista-controlador, que permite separar el tratamiento y el acceso a datos, la lógica de control y la interfaz de usuario. Es un lenguaje multiplataforma, que posee una amplia documentación en su página oficial (http://www.php.net/), entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda. Posee biblioteca nativa de funciones sumamente amplia e incluida, presenta manejo de excepciones e incorpora funciones anónimas y *namespaces*. (5)

Se decide utilizarlo en el desarrollo de la aplicación por brindar las múltiples ventajas mencionadas, además de que es fácilmente integrable con el marco de trabajo seleccionado.

HTML 5

HTML (*HyperText Markup Language*) es un lenguaje abstracto de marcado y estándar para la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código para la definición de contenido en una página web como texto e imágenes. El HTML se escribe en forma de «etiquetas» y además puede incluir o hacer referencia a los script (JavaScript) los cuales pueden afectar el comportamiento de navegadores web y otros procesadores de HTML. Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de la misma manera en cualquier navegador de cualquier sistema operativo. Incorpora etiquetas (canvas 2D y 3D, audio, video) con *codecs* para

mostrar los contenidos multimedia, etiquetas para manejar grandes conjuntos de datos: *Datagrid, Details, Menu y Command*. Permiten generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en cliente. Presenta mejoras en los formularios ya que posee nuevos tipos de datos (*eMail, number, url, datetime*) y facilidades para validar el contenido sin javascript. Añade etiquetas para manejar la web Semántica (Web 3.0): *header, footer, article, nav, time* (fecha del contenido), *link* rel='' (tipo de contenido que se enlaza). (17) .

Es empleado porque el código es más simple que versiones anteriores lo que permite hacer páginas más ligeras que se cargan más rápidamente favoreciendo la usabilidad y la indexación en buscadores.

CSS 3

CSS (*Cascading Style Sheets*) es un lenguaje que se utiliza para definir la presentación de documentos estructurados escritos en HTML con la finalidad de separar la estructura de la presentación y es imprescindible a la hora de crear páginas web complejas. CSS proporciona utilidades para controlar el diseño, espacios, colores, fuentes, y mucho más. (18)

CSS3 introduce nuevos estilos para un mejor control sobre la presentación de las interfaces cada vez más sofisticadas. CSS3 define estilos que controlan la redondez de las esquinas de *divs, spans*, u otros elementos de HTML; así al crear una caja, no requiere de múltiples contenedores anidados e imágenes recortadas, se puede especificar a través de simples definiciones de CSS, en un solo contenedor HTML. Además describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. (18)

Se escoge CSS3 por las ventajas mencionadas en la confección de las plantillas que se mostrarán en la aplicación, permitiendo ahorro de código y tiempo.

Twig 1.12.1

Twig es un lenguaje y motor de plantillas para PHP muy rápido y eficiente, cuyo objetivo es ofrecer una alternativa flexible, potente y segura a otros motores similares que está integrado a Symfony 2 para crear las plantillas de la aplicación, las cuales tienen una sintaxis concisa, fácil de leer y escribir. La característica más importante de Twig es la herencia entre plantillas, lo que permite la creación de un "esqueleto" de plantilla base

que contenga todos los elementos comunes del sitio y define los bloques que las plantillas descendientes pueden sustituir, ahorra considerable tiempo en el trabajo. (19)

Es seleccionado por la integración que presenta con el marco de trabajo escogido, además de presentar características como: rapidez, pues Twig compila las plantillas hasta código PHP regular optimizado; seguro, debido a que tiene un modo de recinto de seguridad para evaluar el código de plantilla que no es confiable, esto permite utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla; y flexible, pues es sustentado por un analizador léxico y sintáctico que permite al desarrollador definir sus propias etiquetas y filtros personalizados, y crear su propio DSL[2].

JavaScript 1.6

JavaScript es un lenguaje de programación interpretado que realiza acciones dentro del ámbito de una página web, fundamentalmente para crear páginas web dinámicas. JavaScript es un lenguaje basado en objetos y orientado a eventos lo que implica que gran parte de la programación en JavaScript se centra en describir objetos (con sus variables de instancia y métodos de "clase") y escribir funciones que respondan a movimientos del cursor, pulsación de teclas, apertura y cerrado de ventanas o carga de una página, entre otros eventos. (20)

Es seleccionado JavaScript ya que permite la ejecución de los programas en el navegador del cliente, sin necesidad de que intervenga el servidor, y es el navegador quien soporta la carga de procesamiento. Posibilita la validación de los datos de los formularios y manejar algunos eventos que son requeridos en las funcionalidades a desarrollar.

1.6.2. Herramienta para el modelado. Visual Paradigm for UML 8.0

Visual Paradigm for UML (VP-UML) es una herramienta CASE[3] profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Es una herramienta multiplataforma que permite una navegación intuitiva entre la escritura del código y su visualización, diseñar los

[2] DSL (Domain Specific Language): el lenguaje específico de dominio es un lenguaje de programación o especificación de un lenguaje dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar una situación particular.
[3] CASE (Computer Aided Software Engineering): aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero.

diversos tipos de diagramas de clases, generar código inverso, código desde diagramas y documentación; además proporciona modelamiento de los requisitos, bases de datos y procesos del negocio. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. (21)

Se seleccionó el *Visual Paradigm for UML* pues además de todas las características antes mencionadas, permite crear de forma rápida y sencilla los diversos diagramas requeridos, posee los recursos necesarios para integrarse con el entorno de desarrollo y el gestor de base de datos escogidos y también que la universidad posee la licencia para su uso.

1.6.3. Aplicación gráfica de gestión de PostgreSQL. PgAdmin III

PgAdmin es una aplicación de diseño y manejo de bases de datos para su uso con PostgreSQL, puede utilizarse para manejar PostgreSQL 7.3 y superiores que se ejecutan sobre casi cualquier plataforma, así como las versiones comerciales y derivados de PostgreSQL como Postgres Plus Advanced Server. PgAdmin es desarrollado por una comunidad de expertos de PostgreSQL en todo el mundo y está disponible en más de una docena de idiomas. Es un software libre publicado bajo la Licencia PostgreSQL. Además está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL[4] sencillas, hasta el desarrollo de bases de datos complejas y es compatible con todas las características de PostgreSQL facilitando así la administración. (22) (23)

PgAdmin III es escogido por tener características que posibilitan que el trabajo con los datos se realice de una manera sencilla y por poseer una gran integración con el gestor de base de datos seleccionado.

1.6.4. Gestor de base de datos. PostgreSQL 9.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[5] y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones se mantiene a la altura de otras bases de datos comerciales. PostgreSQL utiliza un

[4] SQL (structured query language): El lenguaje de consulta estructurado es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.
[5] BSD (Berkeley Software Distribution): Es una licencia de software libre permisiva que permite el uso del código fuente en software no libre. El software modificado puede redistribuirse sin restricciones, como el usuario escoja, libre o propietario.

modelo cliente/servidor y usa multiprocesos en vez de multi-hilos para garantizar la estabilidad del sistema. Además funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. (24)

Por ser una herramienta de código abierto de gran estabilidad, tener una fuerte integración con la herramienta gráfica seleccionada, contar con un gran prestigio y aceptación a nivel internacional y además poseer una gran seguridad en el trabajo con la base de datos, brindando así la confidencialidad de los datos apropiada a las especificaciones del cliente, es seleccionado PostgreSQL como gestor de base de datos para el desarrollo del trabajo.

1.6.5. Entorno de desarrollo integrado (IDE). NetBeans 7.3

NetBeans 7.3 es un entorno de desarrollo de código abierto, es un producto libre y gratuito sin restricciones de uso, una herramienta para escribir, compilar, depurar y ejecutar aplicaciones, está escrito en Java pero puede servir para casi cualquier otro lenguaje de programación. Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web y control de versiones. NetBeans IDE 7.3 permite crear ricas aplicaciones web y móviles utilizando HTML5, JavaScript y CSS3. Posee un amplio soporte para el lenguaje PHP así como para el *framework*[6] de trabajo Symfony 2. (25)

Se elige utilizar NetBeans en su versión 7.3 para el desarrollo de la solución ya que permite la creación y depuración de proyectos PHP, además que provee integración con el SGBD PostgreSQL y con el *framework* Symfony 2 no haciéndose necesario la utilización de una consola externa para trabajar con dicho *framework*. Además de existir una gran experiencia y conocimiento de esta herramienta por el equipo de desarrollo.

1.6.6. Marco de trabajo. Symfony 2.1.7

Symfony 2 es un completo *framework* diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Posee una gran seguridad interna, utilizando técnicas para evitar

[6] Framework (marco de trabajo): es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software

inyección SQL[7], XSS[8] e incluso CSRF[9], así como la integración de PHP-Unit para la realización de pruebas unitarias. Es un marco de trabajo muy documentado, con el que se pueden desarrollar aplicaciones web comerciales, gratuitas y/o de software libre. Ha sido ideado para exprimir al límite todas las nuevas características de PHP 5.3 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 tu proyecto. Symfony está desarrollado completamente en PHP 5.3. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. (26)

Se escoge Symfony 2 ya que está desarrollado con PHP; su propio núcleo está dividido en varios módulos que tienen una alta cohesión, lo que permite que los desarrolladores reutilizar componentes y partes de la aplicación de una manera muy sencilla. Además es soportado por el IDE de desarrollo y compatible con el gestor de base de datos seleccionados previamente.

1.6.7. Mapeo de objeto relacional (ORM). Doctrine 2.3.2

Doctrine es un mapeador de objetos-relacional (ORM) escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un SGBD. (27)

Se utilizó Doctrine debido a que da la opción de escribir las consultas de base de datos en un dialecto SQL propio, llamado Lenguaje de Consulta Doctrine (DQL *Doctrine Query Language*). Permite generar clases a partir de una base de datos existente, presenta también soporte para datos jerárquicos y herencia. Además viene integrado con el marco de trabajo seleccionado Symfony2 por lo que proporciona flexibilidad y agilidad al trabajo.

1.6.8. Servidor web. Apache 2.2.22

Apache es un servidor web HTTP de código abierto para plataformas Unix (BSD, GNU/Linux), Microsoft Windows, Macintosh y otras que implementa el protocolo HTTP.

[7] Inyección SQL: es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos.
[8] Cross-site scripting: es un tipo de agujero de seguridad que permite a una tercera parte inyectar en páginas web vistas por el usuario código en lenguaje script evitando medidas de control.
[9] Cross-site request forgery: es un tipo de exploit malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía.

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido. Tiene la capacidad de servir páginas tanto de contenido estático como dinámico. Facilita la integración de plugins de los lenguajes de programación de páginas web dinámicas más comunes. Tiene integración en estándar del protocolo de seguridad SSL y provee interfaz a todas las bases de datos. (28)

Por las características antes mencionadas, por la necesidad de tener un servidor compatible con la plataforma de desarrollo seleccionada, por brindar rapidez en la navegación y seguridad de los datos con los que se trabaja es seleccionado Apache 2.2.22 como el servidor web.

1.6.9. Control de versiones. Subversion 2.1.7

Es un sistema de control de versiones de código abierto. Subversion ha tenido una amplia adopción, tanto en el campo de código abierto como en el mundo empresarial. Es desarrollado como un proyecto de la Apache Software Foundation, y como tal, forma parte de una rica comunidad de desarrolladores y usuarios. (29)

Se utilizó Subversion pues es de gran utilidad por la simplicidad en el uso de su modelo, y su capacidad para apoyar las necesidades de una amplia variedad de usuarios y proyectos, desde particulares hasta operaciones empresariales a gran escala. Garantizó la atomicidad en las actualizaciones. Facilitó las acciones de añadir, eliminar, modificar ficheros y directorios, minimizando el riesgo de aparición de inconsistencias.

1.7. Patrones

En la tecnología de objetos, un **patrón** es una descripción de un problema y la solución, a la que se da un nombre, que se puede aplicar a nuevos contextos; idealmente, proporciona consejos sobre el modo de aplicarlo en varias circunstancias, y considera los puntos fuertes y compromisos. Muchos patrones proporcionan guías sobre el modo en el que deberían asignarse las responsabilidades a los objetos, dada una categoría específica del problema. (30)

1.7.1. Patrones de diseño

Los patrones de diseño tienen como objetivo fundamental, el diseño de código, de sistemas, de análisis, para lograr facilitar la reutilización de las clases y del propio diseño. (31)

Patrones GRASP (Patrones Generales de Software para Asignación de Responsabilidades)

- **Experto en Información:**

La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De esta forma se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada. (30)

- **Creador:**

¿Quién debería ser el responsable de la creación?

Asignar a la clase B la responsabilidad de crear una instancia de clase A si se cumple uno o más de los casos siguientes:

- *B agrega* objetos de A.
- *B contiene* objetos de A.
- *B registra* instancias de objetos de A.
- *B utiliza más estrechamente* objetos de A.
- *B tiene los datos de inicialización* que se pasarán a un objeto de A cuando sea creado (por tanto, B es un Experto con respecto a la creación de A). (30)

- **Bajo Acoplamiento:**

El **acoplamiento** es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Un elemento con bajo (o débil) acoplamiento no depende de demasiados otros elementos; "demasiados" depende del contexto. Estos elementos pueden ser clases, subsistemas, sistemas.

El patrón **Bajo Acoplamiento** impulsa la asignación de responsabilidades de manera que su localización no incremente el acoplamiento hasta un nivel que nos lleve a los resultados negativos que puede producir un acoplamiento alto.
El **Bajo Acoplamiento** soporta el diseño de clases que son más independientes, lo que reduce el impacto del cambio. No se puede considerar de manera aislada a otros patrones como el Experto o el de Alta Cohesión, sino que necesita incluirse como uno de los diferentes principios de diseño que influyen en una elección al asignar una responsabilidad. (30)

- **Alta Cohesión:**

En cuanto al diseño de objetos, la cohesión (o de manera más específica, la cohesión funcional) es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión. Estos elementos pueden ser clases, subsistemas, entre otros.
Una clase tiene una responsabilidad moderada en un área funcional y colabora con otras clases para llevar a cabo las tareas. (30)

- **Controlador:**

Un **Controlador** es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. Un Controlador define el método para la operación del sistema. 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. El patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento. Un controlador debe delegar en otros objetos el trabajo que se necesita hacer, sólo coordina y controla la actividad (compuesta de varias tareas). (30)

Patrones 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. (32)

1.7.2. Patrones de arquitectura:

El **Modelo Vista Controlador (MVC)** es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. El patrón se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

- El **Modelo**: es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la "vista" aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al "modelo" a través del "controlador".
- El **Controlador**: responde a eventos (usualmente acciones del usuario) e invoca peticiones al "modelo" cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su "vista" asociada si se solicita un cambio en la forma en que se presenta de "modelo" (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el "controlador" hace de intermediario entre la "vista" y el "modelo".
- La **Vista**: presenta el modelo (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho modelo la información que debe representar como salida. (33)

1.8. Métricas

La medición es un elemento clave en cualquier proceso de ingeniería. Las medidas se emplean para una mejor comprensión de los atributos de los modelos que se crean y la evaluación de la calidad de los productos de la ingeniería o de los sistemas que se construyen. Aunque las métricas para software no suelen ser absolutas, proporcionan una manera sistemática de evaluar la calidad a partir de un conjunto de reglas definidas

con claridad. Esto permite al ingeniero descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos. (34)

Métricas para Diseño:

Para medir el diseño se utilizaron dos métricas fundamentales Acoplamiento entre clases (AECO) propuesta por Chidamber y Kemerer y Tamaño de la clase (TC) propuesta por Lorenz y Kidd.

AECO:

Es el número de clases a las cuales una clase está ligada, determinado por el número total de asociaciones de uso (umbral). A medida que AECO aumenta es probable que disminuya la facilidad de reutilización de una clase. Valores elevados de AECO también complican las modificaciones y la prueba que asegura que esas modificaciones se hayan hecho. En general, para cada clase deben mantenerse los valores de AECO más bajos razonables. Esto es consistente con la directriz general para reducir el acoplamiento en el software convencional. (35)

TC:

El tamaño general de una clase se determinar por el número total de procedimientos (umbral) que encapsulan las medidas siguientes:

-El número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.

-El número de atributos (tanto atributos heredados como atributos privados de la instancia) que están encapsulados en la clase. (36)

Si existen valores grandes de TC éstos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilización de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente. (37)

Clasificación	Criterio
Pequeño	$Umbral \leq 20$
Mediano	$20 < Umbral > 30$
Grande	$Umbral \geq 30$

Tabla 1: Clasificación de las clases según umbral

1.9. Pruebas

El desarrollo de Sistemas de software implica la realización de una serie de actividades predispuestas a incorporar errores ya sea en la etapa de definición de requerimientos, de diseño o en el desarrollo. Con el objetivo de garantizar la calidad del software, detectar y corregir estos errores es necesario hacer uso de las pruebas de software. (38) El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos. (39)

Objetivos de las pruebas:

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces. (40)

El objetivo es diseñar casos de prueba que, sistemáticamente, saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y de esfuerzo. La prueba no puede asegurar la ausencia de errores; sólo puede demostrar que existen defectos en el software. (39)

Cualquier producto de ingeniería se puede probar de dos formas:
• **Pruebas de caja negra:** realizar pruebas de forma que se compruebe que cada función es operativa. (39)
• **Pruebas de caja blanca:** desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada. (39)

1.9.1. Pruebas de caja blanca

La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba. (41)

Las pruebas de caja blanca intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.

- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Algunos de los métodos de prueba de caja blanca son los siguientes:

- **Prueba del camino básico:** El método del camino básico permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez. (41)
- **Prueba de bucles:** Es un método que se centra en la validez de la construcción de los bucles. (41)
- **Pruebas de condición:** Es un método de diseño de caso de prueba que ejercita las condiciones lógicas contenidas dentro de un módulo de un programa. (41)
- **Prueba de flujo de datos:** El método selecciona rutas de prueba de un programa de acuerdo con las ubicaciones de las definiciones y el uso de las variables en dicho programa. (41)

1.9.2. Pruebas de caja negra

Las pruebas de caja negra, también denominadas, pruebas de comportamiento, se concentran en los requisitos funcionales del software, es decir, permiten derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa. Las pruebas de caja negra ignoran a propósito las estructuras de control y centran su atención en el dominio de la información para encontrar errores como: funciones incorrectas o faltantes, errores de interfaz, errores en estructuras de datos o en el acceso a base de datos externas, errores de comportamiento o desempeño y errores de inicialización y término. (41)

La validación del software se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos funcionales, que se alcanzan todos los requisitos de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requisitos (por ejemplo, portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento). (41)

1.10 Conclusiones parciales

- En las FGR se realiza el proceso de verificación fiscal el cual presenta problemas de control y celeridad debido a que se realizan manualmente.
- Luego de realizar un minucioso estudio de las aplicaciones de gestión fiscal a nivel internacional se concluye que es necesaria la realización de un sistema nacional que lleve a cabo el proceso de verificación fiscal, pues las existentes no cumplen con las necesidades de la FGR.
- La selección de la metodología de desarrollo RUP con el programa de mejoras llevadas a cabo por el centro CEIGE sirve de gran ayuda en la planificación y desarrollo de la aplicación, haciendo énfasis en la generación de artefactos bien documentados.
- La utilización de las herramientas y lenguajes permiten lograr los objetivos trazados a través del aumento de la productividad, la calidad y la reducción del tiempo de desarrollo.
- Los patrones, métricas y pruebas permiten incrementar la facilidad del desarrollo, la reutilización y mejorar la calidad de la solución.

Capítulo II. Requisitos y Diseño

2.1. Introducción

En el presente capítulo se hace referencia a la especificación de los requisitos funcionales que abarcan los procesos a desarrollar, teniendo en cuenta las restricciones del sistema. Se presenta el diseño realizado que abarca artefactos como son los diagramas de clases, diagramas de secuencia, así como interfaces o vistas de los procesos a desarrollar en el módulo de VF para lograr una representación de la arquitectura, siguiendo los patrones identificados.

2.2. Requisitos

La ingeniería de requisitos proporciona los mecanismos apropiados para entender lo que el cliente quiere, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación, y administrar los requisitos conforme estos se transforman en un sistema operacional. Además debe adaptarse a las necesidades del proceso, el proyecto, el producto y las personas que realizan el trabajo, definiendo lo que el sistema debe hacer, a través de la identificación de las funcionalidades requeridas y las restricciones que se imponen. (41)

Mediante el empleo de técnicas de obtención de requisitos como la entrevista con el cliente y la tormenta de ideas, se definen previamente al trabajo, los requisitos que debe cumplir el sistema, dejando documentado los requisitos funcionales y no funcionales, su validación, y la especificación de cada uno de estos. Algunos de estos resultados se presentarán en el epígrafe.

2.2.1. Requisitos funcionales

Los requisitos funcionales definen las funciones del software o sus componentes sin tomar en cuenta ningún tipo de restricción física y describen un conjunto de entradas, comportamientos y salidas de sistema.

Se identificaron un total de 59 requisitos funcionales:

- RF_VF_VF_191 Listar solicitudes de prórroga de tramitación.
- RF_VF_VF_192 Adicionar solicitud de prórroga de tramitación.
- RF_VF_VF_193 Actualizar solicitudes de prórroga de tramitación.
- RF_VF_VF_194 Vista previa de la solicitud de prórroga de tramitación.

- RF_VF_VF_195 Adicionar constancia de aprobación de prórroga de tramitación.
- RF_VF_VF_196 Listar informes de remisión de la solicitud de prórroga de tramitación.
- RF_VF_VF_197 Adicionar informe de remisión de la solicitud de prórroga de tramitación.
- RF_VF_VF_198 Actualizar informe de remisión de la solicitud de prórroga de tramitación.
- RF_VF_VF_199 Vista previa informe de remisión de aceptación de la solicitud de prórroga de tramitación.
- RF_VF_VF_200 Vista previa informe de remisión de denegación de la solicitud de prórroga de tramitación.
- RF_VF_VF_201 Listar informes de disposición sobre la solicitud de prórroga de tramitación.
- RF_VF_VF_202 Adicionar informe de disposición sobre la solicitud de prórroga de tramitación.
- RF_VF_VF_203 Actualizar informe de disposición sobre la solicitud de prórroga de tramitación.
- RF_VF_VF_204 Vista previa de la aceptación de la solicitud de prórroga de tramitación.
- RF_VF_VF_205 Vista previa de la denegación de la solicitud de prórroga de tramitación.
- RF_VF_VF_206 Listar actas de reunión previa.
- RF_VF_VF_207 Adicionar acta de reunión previa.
- RF_VF_VF_208 Actualizar acta de reunión previa.
- RF_VF_VF_209 Vista Previa acta de reunión previa.
- RF_VF_VF_210 Adicionar constancia de firma del acta de reunión previa.
- RF_VF_VF_211 Listar actas de reunión de conclusiones.
- RF_VF_VF_212 Adicionar acta de reunión de conclusiones.
- RF_VF_VF_213 Actualizar acta de reunión de conclusiones.
- RF_VF_VF_214 Vista previa acta de reunión de conclusiones.
- RF_VF_VF_215 Adicionar constancia de firma del acta de reunión de conclusiones.
- RF_VF_VF_216 Listar actas de reunión de solución de discrepancias.
- RF_VF_VF_217 Adicionar acta de reunión de solución de discrepancias.
- RF_VF_VF_218 Actualizar acta de reunión de solución de discrepancias.
- RF_VF_VF_219 Vista previa del acta de reunión de solución de discrepancias.

- RF_VF_VF_220 Adicionar constancia de firma del acta de reunión de solución de discrepancias.
- RF_VF_VF_221 Adicionar informe de solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_222 Actualizar informe de solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_223 Vista previa de la solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_224 Listar informes de disposición sobre la solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_225 Adicionar disposición sobre la solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_226 Actualizar disposición sobre la solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_227 Vista previa de la denegación de la solicitud de prórroga al término de la solución de discrepancias.
- RF_VF_VF_228 Vista previa de la aprobación de la solicitud de prórroga al término de la solución de discrepancias
- RF_VF_VF_229 Listar informes de resultados.
- RF_VF_VF_230 Adicionar informe de resultados.
- RF_VF_VF_231 Actualizar informe de resultados.
- RF_VF_VF_232 Vista previa del informe de resultados.
- RF_VF_VF_233 Generar anexo 1.
- RF_VF_VF_234 Generar anexo 2.
- RF_VF_VF_350 Listar informes resumen mensual de HDD.
- RF_VF_VF_351 Adicionar informe resumen mensual de HDD ejecutadas por la provincia.
- RF_VF_VF_352 Actualizar informe resumen mensual de HDD.
- RF_VF_VF_353 Eliminar informe resumen mensual de HDD.
- RF_VF_VF_354 Vista previa informe resumen mensual de HDD.
- RF_VF_VF_355 Listar registros de radicación de HPD de la provincia.
- RF_VF_VF_356 Adicionar registro de radicación de HDD de la provincia.
- RF_VF_VF_357 Actualizar registro de radicación de HDD de la provincia.
- RF_VF_VF_358 Eliminar registro de radicación de HDD de la provincia.
- RF_VF_VF_359 Vista previa registro de radicación de HDD de la provincia.
- RF_VF_VF_360 Listar registros de radicación de HDD del municipio.

- RF_VF_VF_361 Adicionar registro de radicación de HDD del municipio.
- RF_VF_VF_362 Actualizar registro de radicación de HPD del municipio.
- RF_VF_VF_363 Eliminar registro de radicación de HPD del municipio.
- RF_VF_VF_364 Vista previa registro de radicación de HDD del municipio.

2.2.2. Requisitos no funcionales

Se definen las diversas características que debe cumplir el sistema basado en las propiedades no funcionales del mismo como la fiabilidad, plataforma de desarrollo, uso de memoria, rendimiento, velocidad, entre otras.

Los requisitos no funcionales que debe cumplir el sistema son genéricos para todo el proyecto y se encuentran englobados en 5 requisitos de confidencialidad, 3 de usabilidad, 3 de restricciones de diseño, 3 de interfaz de usuario, 2 de eficiencia, 2 de soporte, y 1 de documentación; para un total de 19 requisitos no funcionales. Estos se encuentran recogidos en el documento CEGEL_SIGEFII_0113_ERS, sección 3.2 que se localiza en el expediente del proyecto.

2.3. Diseño

2.3.1. Patrones empleados

Patrón de arquitectura MVC:

El patrón arquitectónico empleado en el desarrollo de la investigación es el Modelo-Vista-Controlador, facilitado por el marco de trabajo seleccionado (Symfony2). Las tres capas que lo definen son empleadas en la aplicación de la siguiente manera:

- El **modelo** representa la información o las clases con las que trabaja la aplicación. En la aplicación son todas las clases Repository como AccionRepository, las clases gestoras como AccionGtr las cuales separan la lógica del negocio del resto de la aplicación, facilitando la reutilización de código y las clases entidades que se encuentran en el directorio **Entity** de cada Bundle.
- La **vista** es con la que el usuario interactúa y muestra un aspecto del modelo. En la aplicación son todas las vistas gestionadas por el motor de plantillas Twig, ejemplo gestionarProrrogaTramitacion.html.twig, las cuales son almacenadas en el directorio **Resource/Views**.
- El **controlador** es la pieza de código que se encarga de hacer las peticiones al modelo y obtener los datos que le pasa a la vista para que se las muestre al cliente. En la aplicación todas las solicitudes son gestionadas por un controlador frontal (FrontalController). Estos controladores delegan la verdadera tarea a las

acciones (salvarActualizarProrrogaTramitacionAction), estas acciones son agrupadas en controladores (AccionController) dentro del directorio **Controller.**

Patrones GRASP:

- **Experto:** se utiliza principalmente en el trabajo con las clases que poseen responsabilidades específicas, de acuerdo con la información que manejan. Se evidencia el patrón por ejemplo, en las clases AccionGtr, y las entidades (ProrrogaVF) las cuales poseen toda la información y operaciones relacionadas con las acciones que gestionan.
- **Creador:** se emplea en la clase AccionGtr. En esta clase se encuentran las acciones definidas para las operaciones lógicas del negocio y se ejecutan cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades (ej: ProrrogaVF), por lo que la clase AccionGtr es "creadora" de las entidades.
- **Controlador:** en Symfony2 todas las peticiones web son manipuladas por un controlador frontal (app.php, app_dev.php), que es el punto de entrada único para toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre del módulo con la URL entrada por el usuario. Dicho patrón se refleja en el diagrama de componentes.
- **Alta Cohesión:** se encuentra evidenciado fundamentalmente en las clases controladoras como AccionController, que tienen la responsabilidad de definir y realizar todas las acciones que son solicitadas, paralelo a esto colabora con otras para realizar diferentes operaciones, es decir, es formada por diferentes funcionalidades que están relacionadas entre sí, teniendo un sentido común y un propósito único, proporcionando flexibilidad en el sistema.
- **Bajo Acoplamiento:** consiste en tener relacionadas las clases lo menos posible, pues en caso de realizarse algún cambio, tenga la menor repercusión posible en las demás. Esto permite potenciar la reutilización de código y disminuir la dependencia entre las diferentes clases. Esto se evidencia en todas las clases controladoras que heredan de ProcesoController para lograr un bajo acoplamiento entre ellas y así crear un sistema flexible a grandes cambios.

Patrones GoF

En la categoría de Estructurales:

- **Fachada:** es una interfaz que participa como mediadora entre dos capas definidas de la aplicación. El patrón es utilizado para brindar a la capa de presentación a través del AccionGtr de los métodos que esta necesita, sin tener que comunicarla directamente con el modelo.
- **Decorador:** Añade funcionalidad a una clase dinámicamente, se aplica en la generación de las vistas, añadiendo elementos comunes y reutilizables para varias plantillas como el menú o el pie de página. Un ejemplo de esto es el archivo layoutVF.html.twig que contiene el código común para todas las demás páginas, las cuales al crearse heredan de esta y así se decoran todas las páginas del módulo.

En la categoría de Creacionales:

- **Fábrica abstracta:** define una interfaz para crear un objeto, pero delega la responsabilidad de instanciarlos a sus subclases, promoviendo el encapsulamiento de las partes más variables del sistema. Ejemplo de esto se evidencia en la creación de formularios en las clases controladoras (AccionController), mediante el método createform().

2.3.2. Estándares de codificación

Con el objetivo de lograr unanimidad a lo largo del desarrollo de la aplicación se emplean para la fase de diseño diversos estándares de codificación, algunos de los cuales se muestran a continuación:

Clases:

- Los nombres deben ser descriptivos y concisos. No usar grandes frases o pequeñas abreviaciones.
- Los nombres de las páginas twig serán siempre en minúscula y usar guión bajo para las palabras compuestas. Eje: adicionar_prorroga.
- Los nombres de las clases en mayúscula (*case*). Eje: ProrrogaTramitacion.
- Los atributos siempre en minúscula y usar guión bajo para las palabras compuestas. Eje: persona_proceso.

- Las funciones deben comenzar en minúscula y luego continuar con mayúscula (*lowerCameCase*). Eje: gestionarSolictudProrrogaAction().
- Cada atributo debe tener activo sus getter y setter, además de los generados producto de relaciones. También deben incluir la visibilidad y el tipo de dato.

Métodos del controlador:

- Siempre utilizar el nombre **salvarActualizar** para el caso de las páginas que contienen formularios. Se utiliza el mismo método del controlador para mostrar la página (get) que para persistir (post).
- Los que no tengan formularios, un nombre sugerente para la información que se está mostrando. Eje: gestionarProrrogaTramitacionAction().

Demás métodos:

- Usar el comienzo del nombre **cargar** para los métodos que cargan por Id. Eje: cargarDocumento().
- Usar el comienzo del nombre **listar** para los métodos que llenan la información de los grid. Eje: listarSolicitudProrroga().

2.3.3. Artefactos generados

Mediante el uso de la herramienta Visual Paradigm y el lenguaje UML anteriormente descritos se generaron los artefactos de diseño correspondientes a los procesos de prórroga, reuniones e informes del Módulo VF. En los diagramas que a continuación se muestran no se reflejan todas las clases con las que se trabajan en dicho módulo, pues están recogidas en el diagrama de clases Arquitectura base del proyecto, el cual contiene todas las clases que son comunes para la aplicación, permitiendo su reutilización en otros módulos.

Diagrama de clases persistentes:

El diagrama de clases persistentes que se muestra a continuación refleja las diversas clases con las que se interactúa, y recoge toda la información necesaria para desarrollar el proceso en cuestión: ProrrogaTramitacion. Se muestran las clases con sus respectivas relaciones, así como sus atributos y su cardinalidad. Además se destacan en color verde las clases comunes para el módulo, las cuales contienen la mayoría de los atributos genéricos permitiendo facilidades a la hora de la implementación y la reutilización de código.

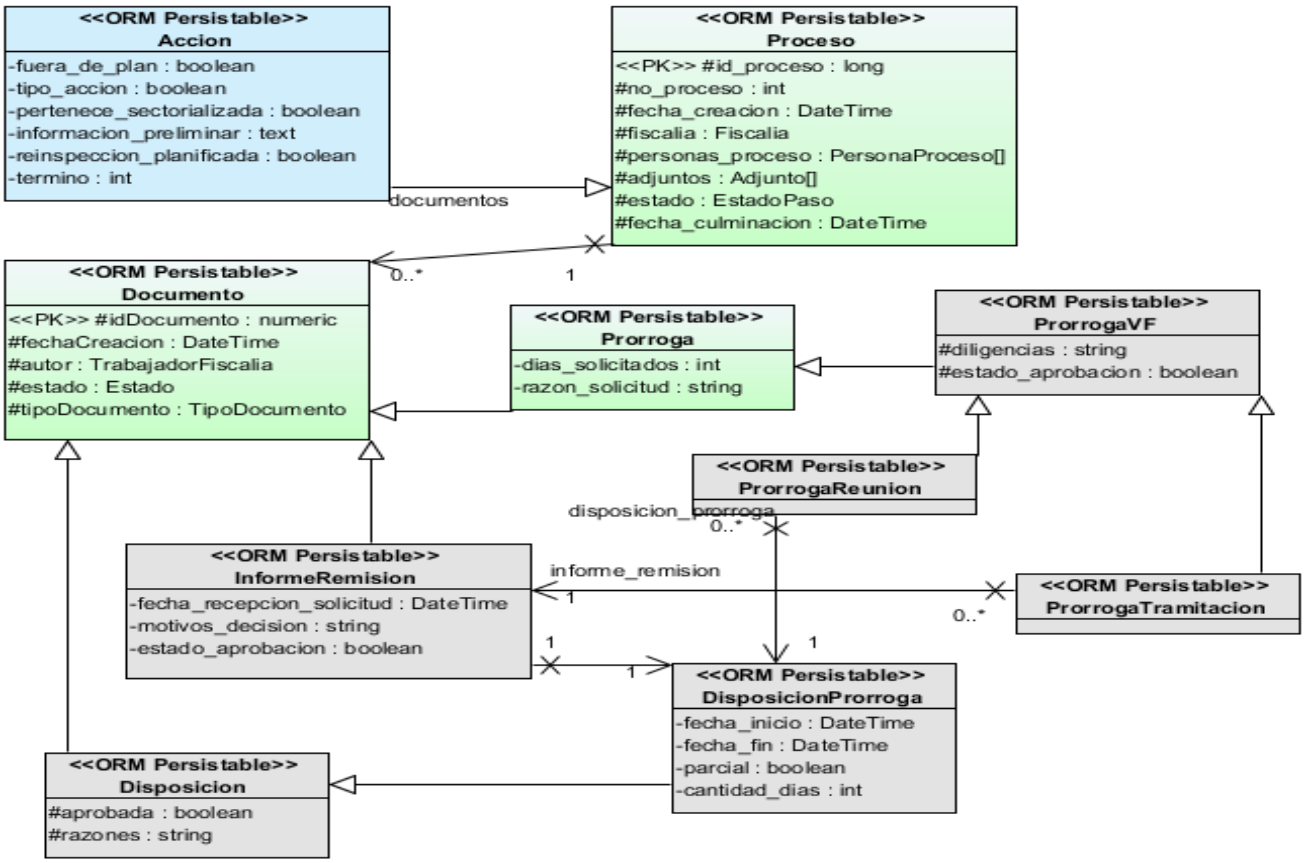


Figura 6: Diagrama de clases persistentes para el proceso Prórroga a la Tramitación.

Fuente: Elaboración propia

Diagrama de clases Controladoras y Gestoras:

Las clases controladoras regulan el trabajo con las clases, encapsulan las acciones o funcionalidades del sistema, coordinando así el trabajo que debe realizar la aplicación. Por su parte las clases gestoras son las intermediarias entre las clases controladoras y las clases repositorio, y se encargan de toda la lógica del negocio. Debido a la estrecha relación que existe entre estos tipos de clases a la hora de satisfacer cualquier funcionalidad del sistema, son incluidas en el mismo diagrama. La siguiente figura representa las clases controladoras (AccionController) y las gestoras (AccionGtr) correspondientes a las funcionalidades pertenecientes al proceso ProrrogaTramitacion en el presente trabajo. Además se evidencia la herencia entre las clases del propio módulo con las clases comunes de la aplicación (ProcesoController y ProcesoGtr).

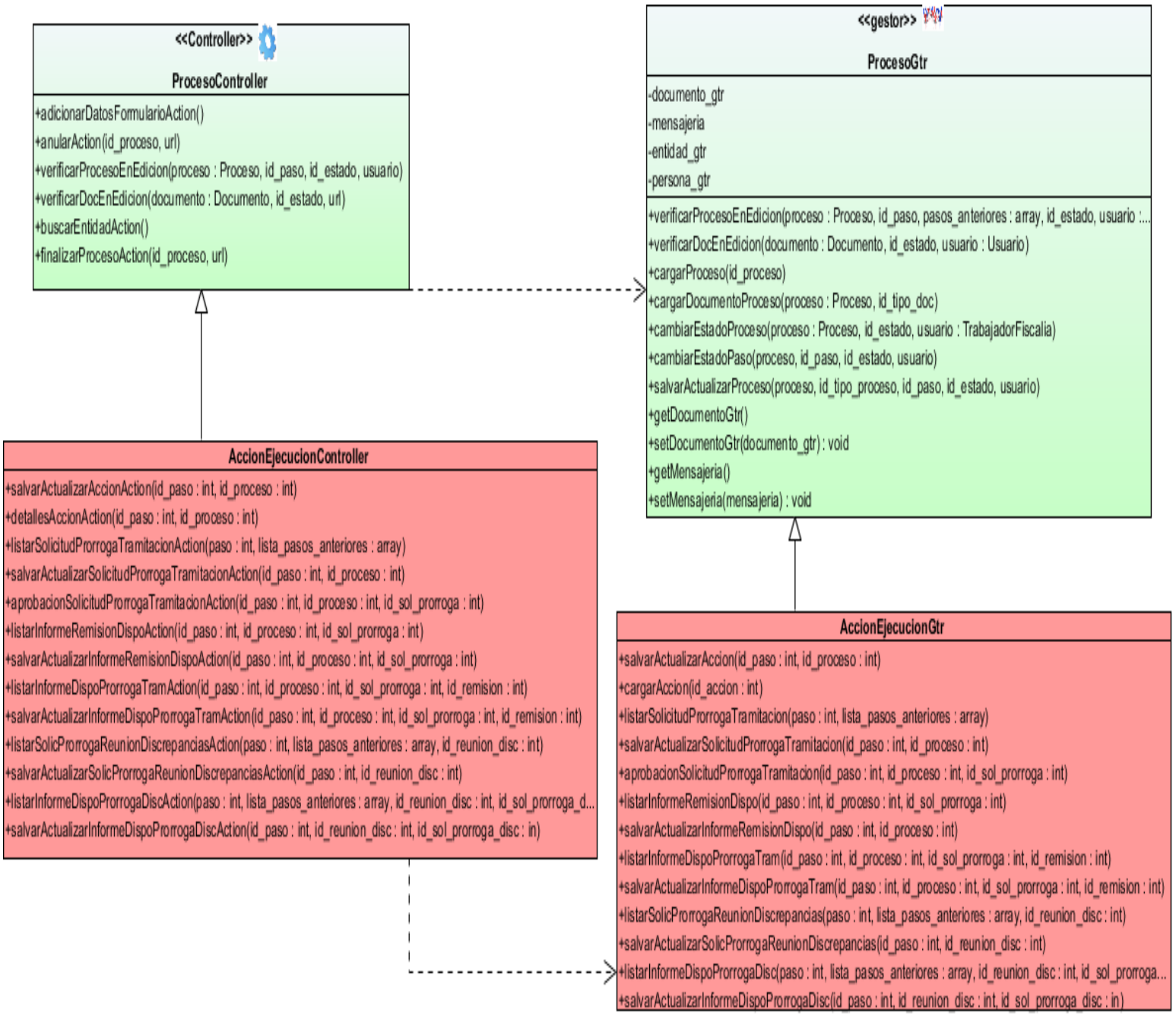


Figura 7: Diagrama de clases controladoras y gestoras para el proceso Prórroga a la Tramitación

Fuente: Elaboración propia

Diagrama de clases de repositorio:

El siguiente diagrama de clases del repositorio permite mostrar las clases encargadas de interactuar con la base de datos, estas contienen todos los métodos necesarios para acceder y obtener la información que requiera la aplicación para su ejecución, ejemplo de estos son: find(), findAll(), findBy() y findByOne(). Además en la figura se muestra la herencia entre la clase común de la aplicación (ProcesoRepository) y la clase propia del módulo (AccionRepository).

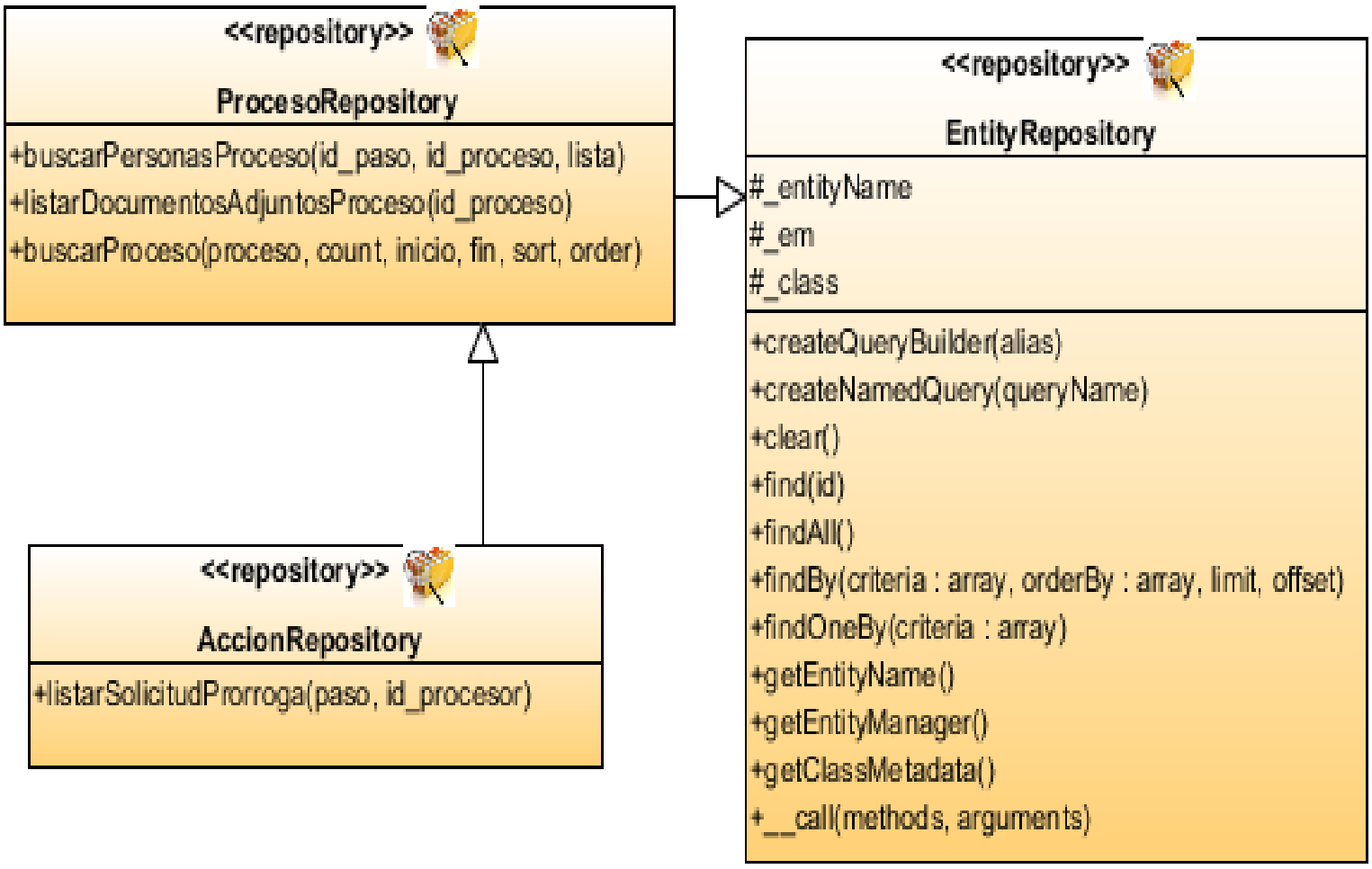


Figura 8: Diagrama de clases de repositorio

Fuente: Elaboración propia

Diagrama de clase de las vistas:

El diagrama de clase de las vistas permitirá una mejor abstracción y comprensión del sistema para su implementación. Dicho diagrama representa todas las páginas de la aplicación incluyendo las vistas previas, los formularios y las vistas generadas o trabajadas con JavaScript.

En la siguiente figura se muestra la navegación básica a través del diagrama de vista para la funcionalidad gestionar_prorroga_tramitacion, perteneciente al proceso de prórrogas en las verificaciones fiscales, las demás funcionalidades del módulo se encuentran en el diagrama de clases Arquitectura base en el expediente de proyecto.

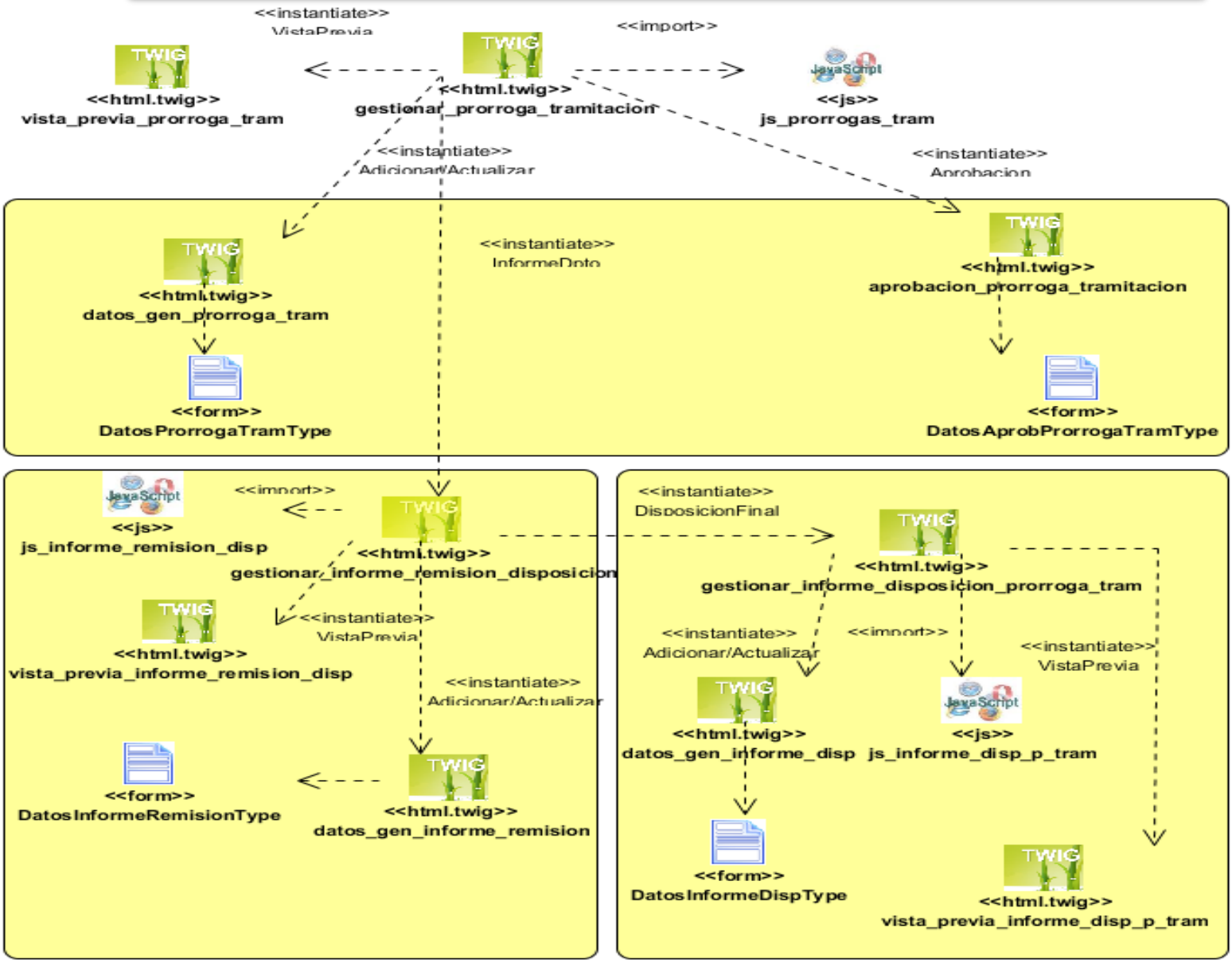


Figura 9: Diagrama de clases de vista para Prórroga a la Tramitación

Fuente: Elaboración propia

Diagrama de secuencia:

El diagrama de secuencia refleja la interacción de los objetos en la aplicación a través del tiempo. Tiene como objetivo describir el comportamiento dinámico del sistema de información, haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos. (42)

El diagrama de secuencia muestra el flujo de pasos que debe realizar el sistema para poder gestionar una prórroga de tramitación.

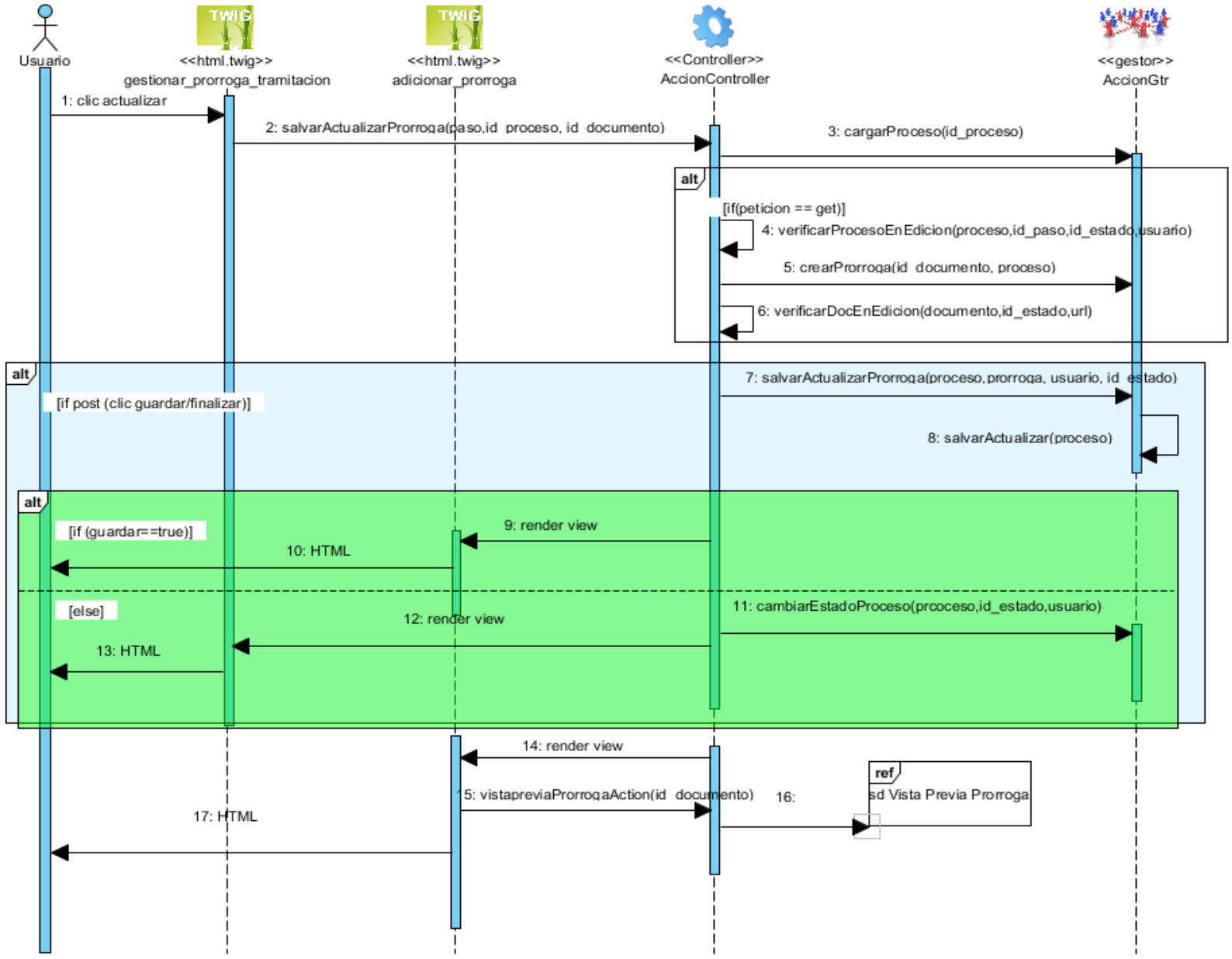


Figura 10: Diagrama de secuencia para el proceso Prórroga a la Tramitación

Fuente: Elaboración propia

2.3.4. Validación del diseño

En el epígrafe se realiza la validación del diseño empleando las métricas reflejadas en el capítulo 1: TC y AECO.

- **TC:**

	Categoría	Criterio
Responsabilidad	Baja	$Umbral \leq Prom$
	Media	$Prom < Umbral > 2 * Prom$
	Alta	$Umbral > 2 * Prom$
Complejidad Implementación	Baja	$Umbral \leq Prom$
	Media	$Prom < Umbral > 2 * Prom$
	Alta	$Umbral > 2 * Prom$
Reutilización	Baja	$Umbral > 2 * Prom$
	Media	$Prom < Umbral > 2 * Prom$
	Alta	$Umbral \leq Prom$

Tabla 2: Clasificación de las clases según la métrica TC

No.	Clases	Umbral	Tamaño	Responsabilidad	Reutilización
1	Documento	46	Grande	Alta	Baja
2	Accion	36	Grande	Alta	Baja
3	Proceso	52	Grande	Alta	Baja
4	PersonaProceso	12	Pequeño	Baja	Alta
5	Datos	6	Pequeño	Baja	Alta
6	Direccion	6	Pequeño	Baja	Alta
7	Prorroga	6	Pequeño	Baja	Alta
8	ProrrogaVF	16	Pequeño	Baja	Alta
9	InformeRemision	13	Pequeño	Baja	Alta
10	Disposicion	6	Pequeño	Baja	Alta
11	DisposicionProrroga	12	Pequeño	Baja	Alta
12	Lugar	12	Pequeño	Baja	Alta
13	TipoLugar	9	Pequeño	Baja	Alta
14	ActaReunion	6	Pequeño	Baja	Alta
15	Inicio	9	Pequeño	Baja	Alta
16	Previa	35	Grande	Media	Media
17	Conclusiones	20	Pequeño	Media	Media
18	RecomprobacionDiscrepancias	19	Pequeño	Media	Media
19	InformeResumen	25	Medio	Media	Media
20	InformeResumenHPD	16	Pequeño	Baja	Alta
21	Provincia	12	Pequeño	Baja	Alta
Prom		17,8			

Tabla 3: Representación del tamaño de clase

Se trabajó con un total de 21 clases, de las cuales se obtuvo como promedio 17.8 de cantidad de procedimientos (umbral).

A continuación se muestran las gráficas que muestran el resultado de medición de los atributos de calidad: responsabilidad, reutilización y complejidad de mantenimiento.

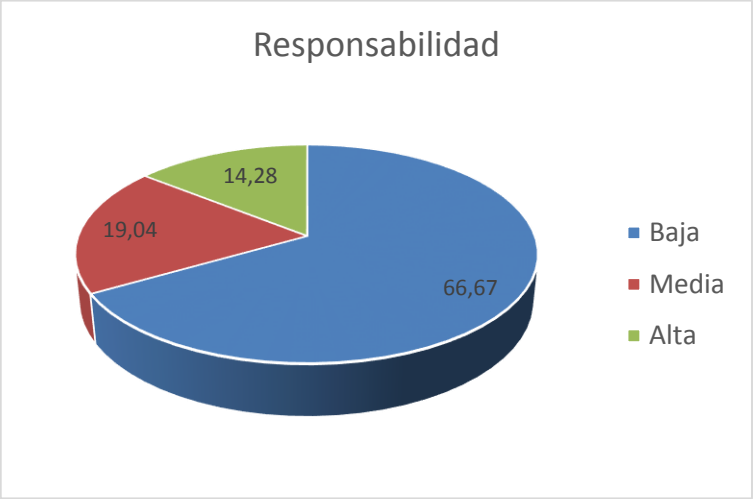


Figura 11: Resultados obtenidos para el atributo Responsabilidad

Fuente: Elaboración propia

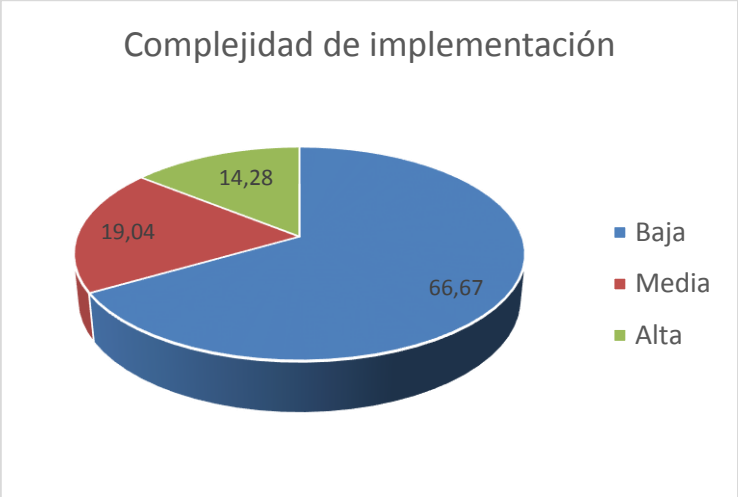


Figura 12: Resultados obtenidos para el atributo Complejidad de implementación

Fuente: Elaboración propia

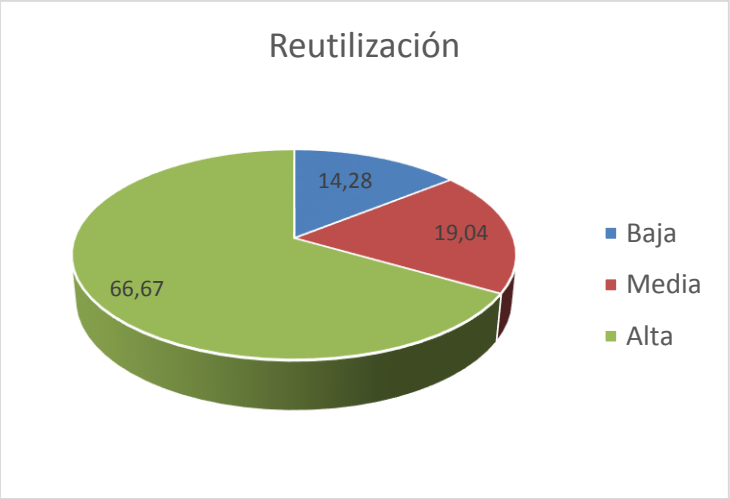


Figura 13: Resultados obtenidos para el atributo Reutilización

Fuente: Elaboración propia

La aplicación de la métrica TC, permitió concluir que las clases del diseño de los procesos de prórroga, reuniones e informes del módulo de VF son pequeñas en su mayoría (16 de 21 clases). No asumen grandes responsabilidades (14 de 21 clases), ni un alto nivel de complejidad de implementación y si cuentan con una alta reutilización de las clases. Por lo que se puede afirmar que los resultados obtenidos en la aplicación de la métrica son positivos.

- **AECO:**

Esta métrica evalúa los siguientes atributos de calidad:

Acoplamiento: responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad del mantenimiento: grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

Reutilización: grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Cantidad de pruebas: número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases) diseñado. (43)

Para los cuales están definidos los siguientes criterios y categorías de evaluación que se muestran en la siguiente tabla:

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	$Umbral = 0$
	Bajo	$Umbral = 1$
	Medio	$Umbral = 2$
	Alto	$Umbral > 2$
Complejidad de mantenimiento	Baja	$Umbral \leq Prom$
	Media	$Prom < Umbral > 2 * Prom$
	Alta	$Umbral > 2 * Prom$
Reutilización	Baja	$Umbral > 2 * Prom$
	Media	$Prom < Umbral > 2 * Prom$
	Alta	$Umbral \leq Prom$
Cantidad de pruebas	Baja	$Umbral \leq Prom$
	Media	$Prom < Umbral > 2 * Prom$
	Alta	$Umbral > 2 * Prom$

Tabla 4: Criterios de evaluación de la métrica AECO.

Al aplicar la métrica se arriba a los siguientes resultados:

No.	Clases	Cantidad de relaciones de uso(umbral)	Acoplamiento	Complejidad Mantenimiento	Reutilización
1	Documento	1	Bajo	Bajo	Alto
2	Accion	1	Bajo	Bajo	Alto
3	Proceso	2	Medio	Medio	Medio
4	PersonaProceso	1	Bajo	Bajo	Alto
5	Datos	0	Ninguno	Bajo	Alto
6	Direccion	0	Ninguno	Bajo	Alto
7	Prorroga	1	Bajo	Bajo	Alto
8	ProrrogaVF	1	Bajo	Bajo	Alto
9	InformeRemision	2	Medio	Medio	Medio
10	Disposicion	1	Bajo	Bajo	Alto
11	DisposicionProrroga	1	Bajo	Bajo	Alto
12	Lugar	2	Medio	Medio	Medio
13	TipoLugar	0	Ninguno	Bajo	Alto
14	ActaReunion	2	Medio	Medio	Medio
15	Inicio	1	Bajo	Bajo	Alto
16	Previa	2	Medio	Medio	Medio

17	Conclusiones	1	Bajo	Bajo	Alto
18	Recomprobacion Discrepancias	2	Medio	Medio	Medio
19	InformeResumen	1	Bajo	Bajo	Alto
20	InformeResumenHPD	1	Bajo	Bajo	Alto
21	Provincia	0	Ninguno	Bajo	Alto
Prom.		1.1			

Tabla 5: Resultados evaluados de la métrica AECO

Para un total de 21 clases y un promedio de asociaciones de uso de 1.1 se obtuvieron los resultados que se muestran en las siguientes figuras:

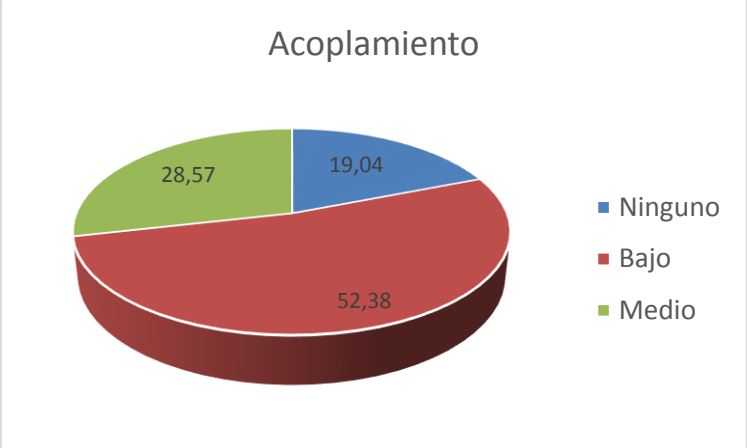


Figura 14: Resultados obtenidos para el atributo Acoplamiento

Fuente: Elaboración propia

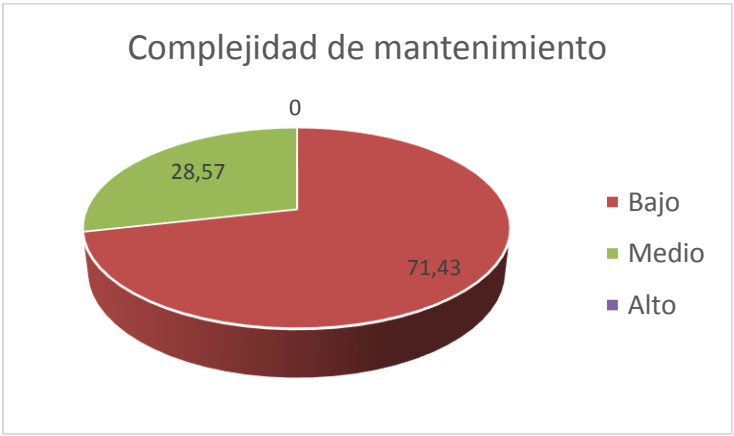


Figura 15: Resultados obtenidos para el atributo Complejidad de mantenimiento

Fuente: Elaboración propia

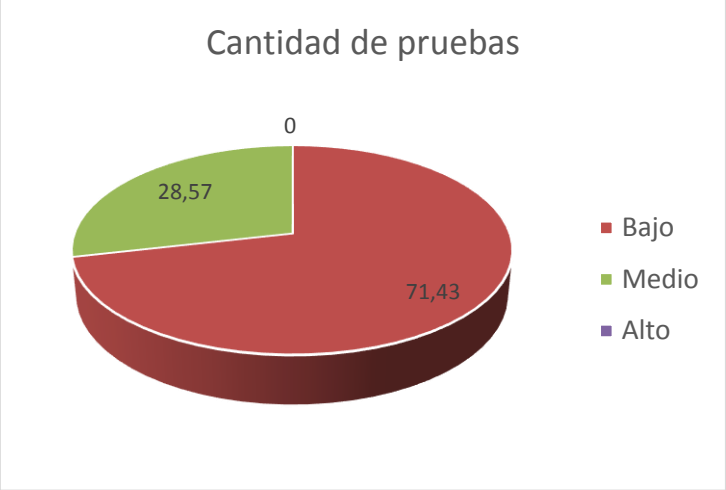


Figura 16: Resultados obtenidos para el atributo Cantidad de pruebas

Fuente: Elaboración propia

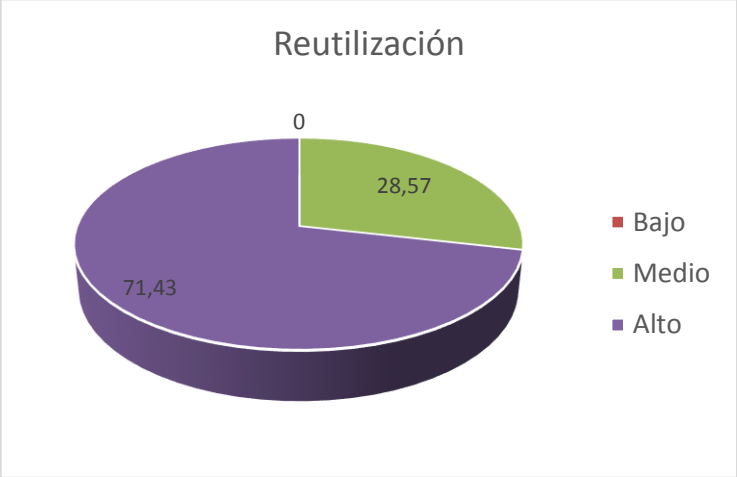


Figura 17: Resultados obtenidos para el atributo Reutilización

Fuente: Elaboración propia

Los resultados arrojados por la métrica empleada permiten determinar y concluir que las clases del diseño presentan un acoplamiento en su mayoría bajo (15 clases de bajo o ningún acoplamiento), arrastrando valores igualmente bajos para los atributos de complejidad de mantenimiento y cantidad de pruebas (15 de 21 clases), derivándose que sea alta la reutilización de las clases, lo que permite determinar que la calidad del diseño realizado es aceptable.

2.4. Conclusiones parciales

Luego de culminado el presente capítulo se llega a las siguientes conclusiones:

- Se definió el uso del patrón de arquitectura MVC el cual separa los datos y la lógica de negocio, de la interfaz de usuario y del módulo encargado de gestionar los eventos y las comunicaciones, facilitando el desarrollo de la aplicación y su posterior mantenimiento.
- La utilización de los patrones de diseño permitió una solución óptima brindando comodidades como son la reutilización de código y la solución a problemas que suelen aparecer en el desarrollo del software.
- La etapa de diseño brindó una visión más profunda de la aplicación. En esta fase se generaron los diagramas de clases persistentes, de repositorio, de controladoras, de la vista y los diagramas de secuencia, facilitando una guía a seguir para la posterior implementación de la aplicación.
- La aplicación de métricas permitió validar el correcto diseño de las clases de la aplicación, basado en variables fundamentales como la responsabilidad y reutilización de las clases.

.

Capítulo 3. Implementación y prueba

3.1. Introducción

Una vez concluido el modelo del diseño, se dispone de los detalles suficientes para proceder a la construcción del sistema, y una vez concluido, se realiza la verificación del cumplimiento de los requisitos funcionales mediante las pruebas de software. En el presente capítulo se construye el modelo de implementación correspondiente al módulo de VF, desglosándolo en los diagramas de componentes de los requisitos más importantes, de la misma forma se determinan los tipos de pruebas a realizar y los casos de pruebas que serán aplicados al sistema.

3.2. Implementación

El modelo de implementación es el artefacto que se genera en esta disciplina. Está conformado por el diagrama de componentes y el diagrama de despliegue, describiendo cómo los elementos del modelo de diseño se implementan en términos de componentes, ficheros de código fuente y ejecutables, además de cómo estos se organizan de acuerdo a los nodos específicos en el diagrama de despliegue. Se describe también cómo se organizarán los componentes de acuerdo con los mecanismos de estructuración, disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y cómo dependen los componentes unos de otros. (41)

3.2.1. Artefactos

Diagrama de componentes

El diagrama de componentes se representa como un grafo de componentes de software unido por medio de las relaciones de dependencia. Entre los componentes se encuentran las clases, interfaces, librerías y componentes ejecutables. Normalmente los diagramas de componentes se utilizan para modelar código fuente, versiones ejecutables, bases de datos físicas entre otros. El diagrama de componentes muestra un conjunto de ficheros relacionados entre sí para lograr una completa funcionalidad del sistema.

Componente:

Un componente es una parte física y reemplazable de un sistema conformado por un conjunto de interfaces y proporciona la realización de dicho conjunto. Se usan para modelar los elementos físicos que pueden hallarse en un nodo por lo que empaquetan elementos como clases, colaboraciones e interfaces. (41)

La siguiente figura muestra el diagrama de componentes del módulo VF. Representa la estructura del proyecto SIGEF II, que contiene el subsistema VF, al cual pertenece dicho módulo. En el subsistema SIGEF II se encuentran archivos de configuración como son: Security, encargado del trabajo de la seguridad del subsistema; archivo Routing, gestiona las rutas del subsistema y el Config, que posee las principales configuraciones de SIGEF II. El componente Ctr_Frontal es el controlador frontal que recibe cada petición realizada al sistema. El paquete Arquitectura Base contiene componentes con funcionalidades comunes para todo el sistema, como son la Seguridad, Servicios y Componentes.

El módulo VF está compuesto por tres paquetes fundamentales, el modelo, la vista y el controlador. En el paquete Controlador se encuentra el componente Controladoras que está formado por todas las clases controladoras correspondientes a los procesos que se llevan a cabo en el módulo, por ejemplo, AccionController, encargadas de dar respuesta a las peticiones de los usuarios. Controladora se relaciona con el componente Negocio, compuesto por las clases gestoras, por ejemplo, AccionGtr, que se ocupan de gestionar la lógica del negocio. Ambos componentes se relacionan con el componente Entidades que pertenece al paquete Modelo, el cual contiene las clases entidades del módulo. Entidades es usado por el componente Repositorio encargado de realizar las consultas a la base de datos. El paquete Vista, contiene el componente Form encargado del trabajo con los formularios, el componente Twig tiene como objetivo el trabajo con las interfaces de usuario y el componente Extensiones que posee los elementos javascript y los css. Routing es el encargado de trabajar con las rutas definidas para el módulo. Services, ofrece los servicios públicos a los que se accede desde cualquier parte del subsistema VF, en el componente se registran las clases gestoras para acceder a ellas desde los controladores. En el subsistema VF se encuentra el componente Mensajería, que gestiona los mensajes que deben ser mostrados y Excepciones para la captura y tratamiento de las excepciones generadas.

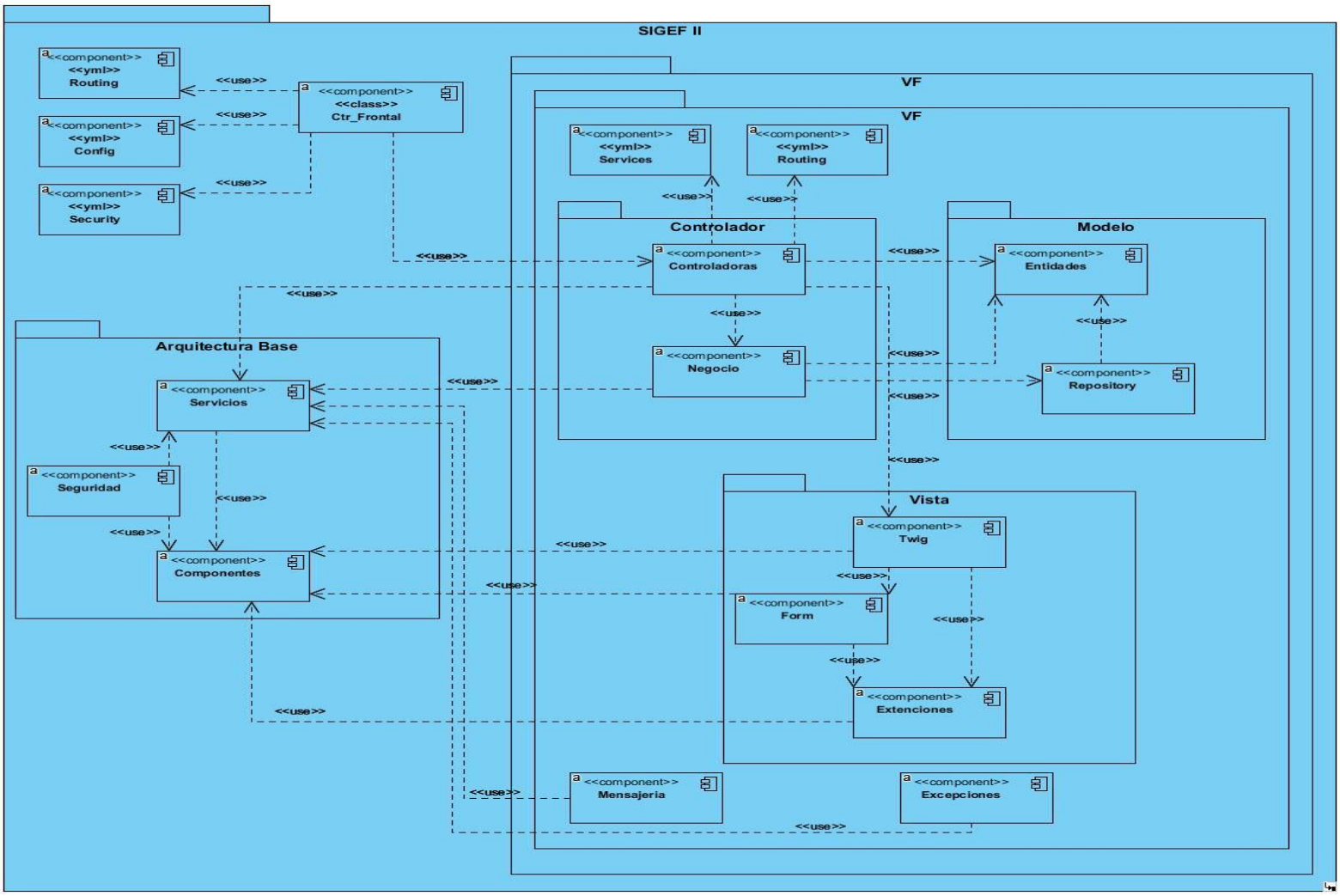


Figura 18: Diagrama de componentes

Fuente: Elaboración propia

Diagrama de despliegue

El diagrama de despliegue describe la distribución física del sistema en términos de cómo las funcionalidades se distribuyen entre los nodos de computación sobre los que se va a instalar el sistema, mostrando las relaciones entre el hardware y el software en el sistema final. Se representa como un grafo de nodos unidos por conexiones de comunicación. (12)

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.

Para el despliegue del sistema se utiliza un servidor web, un servidor de base de datos, y máquinas clientes en cada una de las instancias de la fiscalía; además de un servidor de base de datos en la sede nacional para la realización de las réplicas entre servidores, así como un servidor de copias de seguridad en caso de existencia de algún inconveniente que repercuta en perdida de información. Las máquinas clientes serán clientes ligeros con una distribución de Nova o Ubuntu compatibles con el navegador Firefox 10.X o superior, mientras que los servidores utilizarán como distribución Ubuntu 10.X en versión estable según los recursos disponibles del cliente. La conexión entre servidores de base de datos y los servidores web será mediante el protocolo TCP, mientras que entre el servidor web y las máquinas clientes serán por el protocolo HTTPS.

A continuación se muestra la representación del diagrama de despliegue.

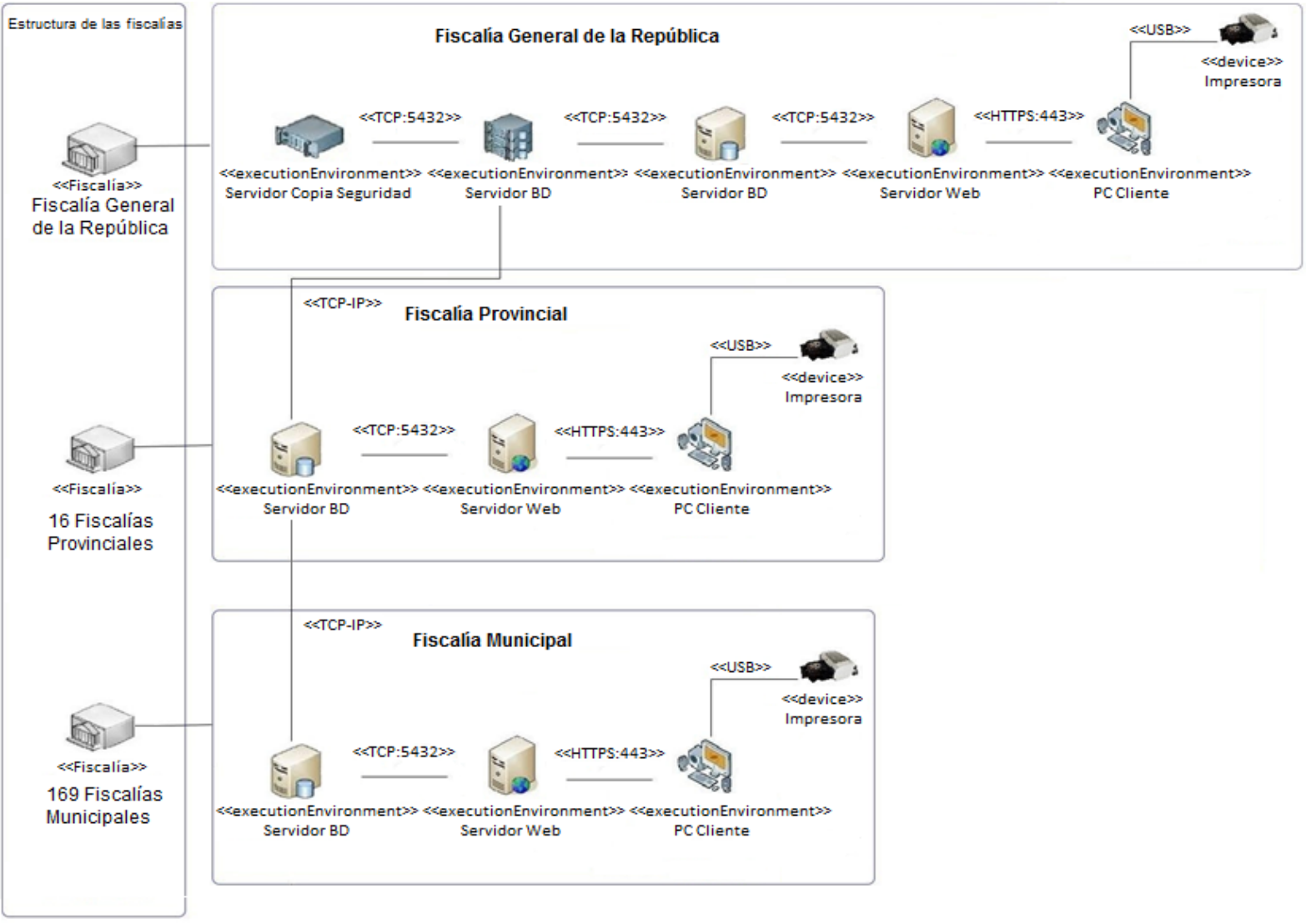


Figura 19: Diagrama de despliegue

Fuente: Elaboración propia

3.2.2. Estándares de codificación

Los estándares de codificación para la implementación fueron definidos por la dirección del proyecto antes de comenzar el desarrollo, para así lograr uniformidad en el código de todos los módulos de la aplicación. Dicho estándar de codificación se encuentra en el documento “Estándares de codificación para PHP” del proyecto SIGEF II.

Funciones y métodos: los nombres de funciones pueden contener únicamente caracteres alfanuméricos. Los guiones bajos (_) no están permitidos. Los números están permitidos en los nombres de función pero no se aconseja en la mayoría de los casos.

Clases abstractas: en general, las clases abstractas siguen las mismas convenciones que las clases, con una regla adicional: Los nombres de las clases abstractas deben acabar con el término, "Abstract", y ese término no debe ser precedida por un guión bajo.

Nombres de archivos: cualquier archivo que contenga código PHP debe terminar con la extensión " .php ", con la excepción de los scripts de la vista.

Clases: los nombres de clases pueden contener sólo caracteres alfanuméricos. Los números están permitidos en los nombres de clase, pero desaconsejados en la mayoría de los casos. Las barras bajas (_) están permitidas solo como separador de ruta (el archivo " Entity/Table.php " debe apuntar al nombre de clase " Entity_Table ").

Cadenas de texto entre comillas: PHP tiene dos formas de poner string o cadenas de texto. Con comillas simples y con comillas dobles. La diferencia es que si se usa comillas dobles y se coloca dentro del texto un nombre de variable, el compilador lo interpretará y reemplazará por su valor. Por esta razón siempre se va a usar comillas simples a menos que se necesite hacer la interpolación de variables que permiten las dobles.

Números dentro del código: convertir en constante un número cuando se necesite. No usar variables sin inicializar. Si no se tiene control sobre el valor de una variable, se verifica que esté inicializada.

3.3. Pruebas

3.3.1. Pruebas de caja blanca

Para realizar la prueba de caja blanca se tomó la funcionalidad Adicionar/Actualizar Prórroga a la Tramitación al cual se le aplica la prueba del camino básico.

```
public function salvarActualizarProrrogaAction($paso, $id_proceso, $id_documento)
    {
        $peticion = $this->getRequest();
        $proceso = $this->getAccionGtr()->obtenerProceso($id_proceso);
        $prorroga = $this->getAccionGtr()->crearProrroga($id_documento, $proceso);
        $resultado = $this->verificarProcesoEnEdicion($proceso, $this->listaPasosAnteriores($paso), $paso, ConfigUtilVF::estado_edicion,
        $this->generateUrl('vf_gestionar_solicitud_prorroga_1', array('paso' => ConfigUtilVF::Prorroga_tramitacion, 'id_proceso' => $id_proceso))
        if (!is_null($resultado)) {
            return $resultado;
        }
        $prorroga = $this->getAccionGtr()->cargarDocumentoProrroga($proceso, ConfigUtilVF::doc_prorroga);
        $form = $this->createForm(new SolicitudProrrogaType(), $prorroga, array());
        if ($peticion->getMethod() == 'POST') {
            $form->bind($peticion);
            if ($form->isValid()) {
                $usuario = $this->getUser();
                $tipo_boton = $peticion->get(ConfigUtilVF::btn_finalizar);
                if (isset($tipo_boton)) {
                    $this->getAccionGtr()->salvarActualizarProrroga($proceso, $prorroga, $usuario, ConfigUtilVF::estado_pendiente);
                    $this->getAccionGtr()->cambiarEstadoProceso($proceso, ConfigUtilVF::estado_finalizado, $usuario);
                    $errores[] = 'me_adm:me_finalizar';
                    $this->adicionarFlashError($errores);
                    return $this->redirect($this->generateUrl('vf_gestionar_solicitud_prorroga_1',
                    array('paso' => ConfigUtilVF::Prorroga_tramitacion, 'id_proceso' => $id_proceso)));
                else{
                 $tipo_boton = $peticion->get(ConfigUtilVF::btn_guardar);
                if(isset($tipo_boton)) {
                    $this->getAccionGtr()->salvarActualizarProrroga($proceso, $prorroga, $usuario, ConfigUtilVF::estado_pendiente);
                    $this->getAccionGtr()->cambiarEstadoProceso($proceso, ConfigUtilVF::estado_pendiente, $usuario);
                    $errores[] = 'me_adm:me_guardar';
                    $this->adicionarFlashError($errores);
                    return $this->redirect($this->generateUrl('vf_salvar_actualizar_solicitud_prorroga',
                    array('paso' => ConfigUtilVF::Prorroga_tramitacion, 'id_proceso' => $id_proceso, 'id_documento'=>$id_documento)));
                }
                }
            }
        }
        return $this->render('VFBundle:Accion/ProrrogaTram:adicionar_prorroga.html.twig', array(
            'form' => $form->createView(),
            'paso' => $paso,
            'id_proceso' => $id_proceso,
            'id_documento' => $prorroga->getIdDocumento(),
        ));
    }
```

Figura 20: Método salvarActualizarProrrogaAction

Fuente: Elaboración propia

El primer paso para realizar la prueba es dibujar el grafo del flujo asociado al código en cuestión, las sentencias son representadas de la siguiente forma:

Notación del grafo de flujo o grafo del programa

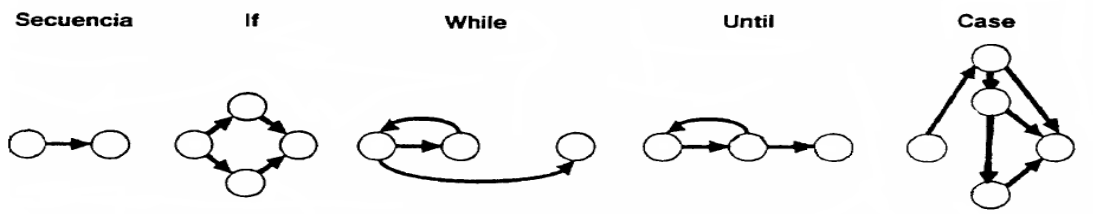


Figura 21: Notación del grafo de flujo

Fuente: Ingeniería de software. Un enfoque práctico

Así queda representado el grafo:

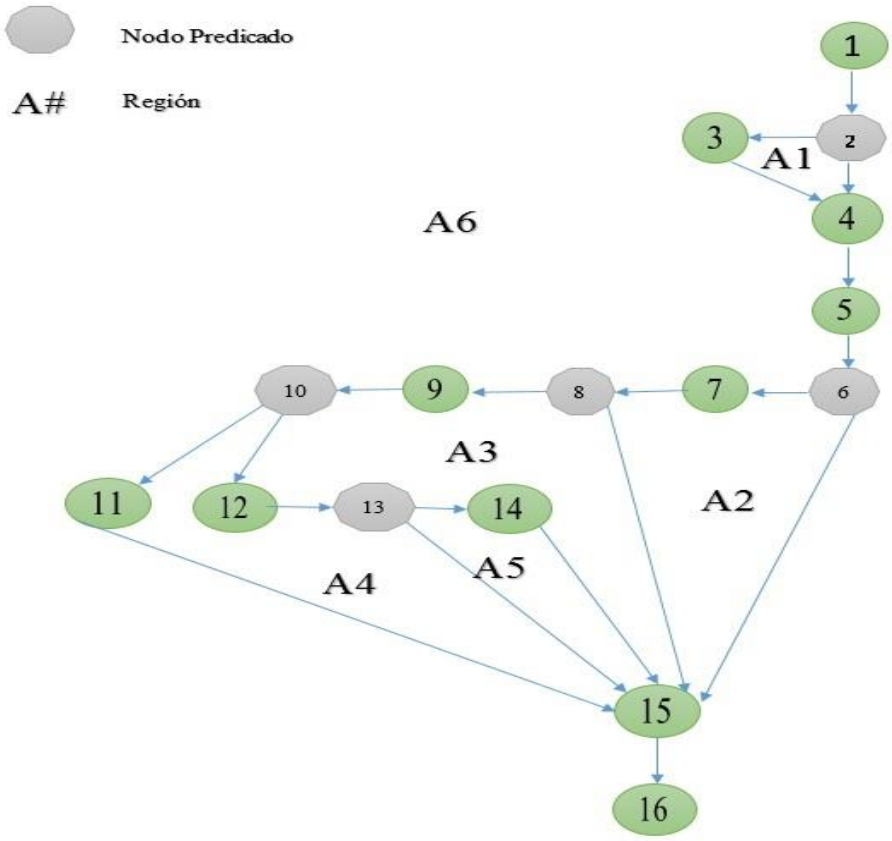


Figura 22: Grafo del camino básico para el método salvarActualizarProrrogaAction

Fuente: Elaboración propia

Complejidad ciclomática:

1. V (G) = 6

2. V (G) = A-N+2 = 20-16+2 = 6

3. V (G) = P+1 = 5+1 = 6

El valor calculado define el número de caminos linealmente independientes del conjunto básico del programa, por lo que se obtienen 6 caminos:

Camino 1: 1-2-4-5-6-15-16

Camino 2: 1-2-3-4-5-6-15-16

Camino 3: 1-2-4-5-6-7-8-15-16

Camino 4: 1-2-3-4-5-6-7-8-15-16

Camino 5: 1-2-4-5-6-7-8-9-10-12-13-15-16

Camino 6: 1-2-3-4-5-6-7-8-9-10-12-13-15-16

Luego de establecidos los caminos básicos se procede a realizar los casos de prueba para casa uno de ellos, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino. A continuación se presenta un ejemplo de los 6 casos de pruebas realizados a esta funcionalidad.

Entrada	Primeramente se obtiene el proceso, luego se crea la prórroga, se verifica que no esté en edición, se carga la prórroga y se crea el formulario. La petición es de tipo GET.
Resultados esperados	Se espera que devuelva la página html adicionar_prorroga_form
Condiciones	$resultado = null, $peticion = GET

Tabla 6: Caso de prueba para camino 1

Una vez culminadas las pruebas del camino básico al método salvarActualizarProrrogaAction se encontraron 2 errores en el código, los cuales fueron erradicados en una segunda iteración de prueba, de las cuales se obtuvieron resultados satisfactorios. Se comprobó que la funcionalidad cumple con todas las condiciones del flujo de trabajo, por lo que se concluye que es correcta.

Como parte de las pruebas de caja blanca aplicadas al módulo de VF, se hizo empleo del *framework* PHPUnit[10]. A continuación se muestran los resultados obtenidos de dichas pruebas en una segunda iteración.

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

	Coverage								
	Lines			Functions / Methods			Classes		
Total		94.66%	2483/2623		97.47%	424 / 435		100.00%	40 / 40
Controller		93.26%	692 / 742		100.00%	63 / 63		100.00%	2 / 2
Form		100.00%	462 / 462		100.00%	47 / 47		100.00%	15 / 15
Negocio		89.94%	438 / 487		100.00%	73 / 73		100.00%	2 / 2
Entity		95.60%	891 / 932		95.63%	241 / 252		100.00%	21 / 21
VFBundle.php		100.00%	1 / 1		100.00%			100.00%	1 / 1

Figura 23. Resultados de la aplicación de pruebas unitarias con PHPUnit

Fuente: Elaboración propia

Después de aplicadas las pruebas a las funcionalidades implementadas se obtuvo como resultado un 94.66% de las líneas de código ejecutadas, un 97.47% de las funcionalidades y un 100% para las clases. Por lo que se puede afirmar que las pruebas evidencian una ejecución correcta del código.

3.3.2. Pruebas de caja negra

Las pruebas de caja negra realizadas durante el proceso de revisión del software pretenden encontrar a través de los casos de prueba los errores que presenta el sistema desarrollado.

La técnica utilizada en la investigación para hacer estas pruebas es la **Técnica de la Partición de Equivalencia**: Técnica que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. La partición equivalente se dirige a una definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. (41)

A continuación se muestra el resultado del caso de prueba para el escenario Adicionar Prórroga a la Tramitación:

[10] **PHPUnit** es un framework para PHP empleado para realizar pruebas unitarias y analizar los resultados obtenidos en ellas.

Escenario	Descripción	Fecha emisión	Cant. Días	Diligencias a ejecutar	Elementos que sustentan el pedido	Fiscal Actuante	Respuesta del sistema	Flujo central
EC 1.1 Adicionar prórroga con datos correctos	Permite adicionar una prórroga a una acción y actualizarlo en el sistema.	V	V	V	V	V	Debe adicionar una prórroga al listado de prórrogas de la acción seleccionada	1. Selecciona la opción Prórroga tramitación del menú vertical. 2. Selecciona la acción a la cual agregarle la prórroga y selecciona Actualizar. 3. Muestra una pantalla con los datos a introducir de la prórroga. 4. Selecciona finalizar
		20/05/2014	3	Diligencia	Elementos	Administrador		
		V	V	V	V	V		
		22/05/2014	8	Algunas diligencias	Algunos elementos	José Pablo		
EC 1.2 Adicionar prórroga con datos incorrectos		V	I	V	V	V	Debe mostrar un mensaje indicando que la cantidad de días no es válido pues el campo solo admite números enteros	
		25/05/2014	So	Dili	Ele	Administrador		
		V	I	V	V	V	Debe mostrar un mensaje indicando que la cantidad de días no es válido pues el máximo de días que admite es 10	
		24/05/2014	88	diligencias	Varios elementos	José Pablo		

EC 1.3 Adicionar prórroga con datos incompletos		I	V	V	V	V	Debe mostrar un mensaje de error de que existen campos obligatorios vacíos	
		-	5	diligencias	elementos	Administrador		
		V	I	V	V	V		
		24/05/2014	-	diligenc	Algún elemento	Administrador		
		V	V	I	V	V		
		25/05/2014	4	-	element	José Pablo		
		V	V	V	I	V		
		24/05/2014	3	diligenia1	-	Administrado		
		V	V	V	V	I		
		25/05/2014	7	Diligencia4	Elemeto3	-		

(*Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, N/A que no es necesario proporcionar un valor del dato, ya que es irrelevante*)

Tabla 7: Caso de prueba de caja negra para el RF Adicionar Prórroga

Al concluir las pruebas de caja negra para la funcionalidad Prórroga a la Tramitación, en una primera iteración se detectaron un total de dos no conformidades, las cuales fueron resueltas. Al aplicar una segunda iteración a esta funcionalidad, se obtuvieron resultados satisfactorios.

3.3.3. Validación de la propuesta solución

Con el objetivo de validar la propuesta solución, se coordinó un encuentro con fiscales en el laboratorio del proyecto SIGEF II, en el cual se obtuvieron datos reales del control y del tiempo de ejecución manual de los procesos de prórrogas, reuniones e informes de VF.

La gestión de la información se mide a través del control que se lleva de los procesos y la celeridad en el tiempo en que se realice cada uno de ellos. A continuación se muestran los resultados obtenidos de la comparación con respecto al comportamiento de las variables de la investigación, control y celeridad:

Dimensiones de la Variables	Antes	Después
Acceso a la información	No se conoce la información del proceso si no se tiene el documento en formato duro.	Contando con los permisos necesarios, se puede acceder a la información del documento desde cualquier instancia de la fiscalía.
Persistencia de datos	Los datos son siempre escritos en el proceso que se esté realizando, sin importar que se conozcan o estén recogidos en un proceso anterior.	Los datos se guardan en la base de datos, desde la cual se podrá acceder a los mismos y trabajar sobre ellos, asociándoles personas o entidades, sin necesidad de adicionarlas nuevamente.
Reportes estadísticos	Los fiscales llenan los datos de la plantilla de reportes con la información encontrada en los libros de control. Esta información está sujeta a errores, teniendo en cuenta que el conteo es manual.	Se realizan fácilmente mediante diversos criterios de búsquedas, mostrando la información real de los procesos en cuestión. Tiene como desventaja que aquellos datos que no se hayan introducido en el sistema no serán contados.

Tabla 8: Validación para la variable control

Dimensiones de la Variables	Antes	Después
Tiempo de demora de las búsquedas de documentos	De 10 a 30 minutos, dependiendo del tiempo que lleva creado y archivado.	De 2 a 8 segundos. El tiempo de respuesta de la aplicación es de hasta 8 segundos para las consultas más grandes.

Tiempo de elaboración de un proceso	De 10 a 20 minutos dependiendo del tamaño del documento.	De 5 a 15 minutos.

Tabla 9: Validación para la variable celeridad

3.4. Conclusiones parciales

Al concluir el presente capítulo se puede afirmar:

- El diagrama de componentes muestra la estructura lógica del módulo VF, basado en el desarrollo de componentes, permitiendo encapsular las funcionalidades y lograr un proceso de implementación más organizado y productivo.
- El modelo de despliegue permite identificar la distribución de los nodos necesarios para el despliegue del sistema a nivel nacional. Las tecnologías representadas favorecen el despliegue de la solución.
- Los resultados de la realización de pruebas de caja negra mediante la técnica partición de equivalencia, evidencian que las funciones del sistema son operativas, que las entradas se aceptan de forma adecuada y que se producen salidas correctas, con esta prueba se detectaron varias no conformidades, a las cuales se les dio solución rápidamente, obteniendo finalmente una aplicación que satisface los requisitos definidos. Mientras que las pruebas de caja blanca realizadas mediante la técnica del camino básico y el uso de la herramienta PHPUnit para la validación del código, reflejan que las operaciones internas se ajustan a las especificaciones y que los componentes internos funcionan de forma adecuada.

Conclusiones Generales:

Con el desarrollo del trabajo de diploma se demostró la necesidad de la implementación del módulo VF, pues se lograron resolver los principales problemas que afectan en la actualidad al área de VF. Luego de terminada la investigación se concluye que:

- Se investigaron profundamente los sistemas existentes en el mundo relacionados con el proceso de verificación fiscal, identificando sus principales características las cuales fueron tomadas en cuenta en la implementación de la solución propuesta.
- Las técnicas, metodologías y patrones empleados en las diferentes fases en la construcción del software, permitieron la correcta y rápida implementación del módulo VF, lo cual se evidenció en los artefactos obtenidos durante el desarrollo.
- El diseño, mediante el uso de patrones y los diagramas generados, permitió implementar los procesos prórrogas, reuniones e informes de forma eficiente.
- La implementación de las funcionalidades sobre la base de la arquitectura del módulo VF, sustentada en componentes, potenció la reutilización del código y la reducción del tiempo de desarrollo.
- Los resultados obtenidos con las pruebas de usuario, demostraron la superioridad de las variables celeridad y control de los procesos ejecutados en la nueva aplicación y los obtenidos de forma manual por los fiscales, dándole solución de esta forma al problema de la investigación.

Recomendaciones:

Se recomienda:

Implementar las restantes funcionalidades del módulo Verificación Fiscal como Inconformidad, Impugnación y Reinspección.

Implementar una funcionalidad para la firma digital de los documentos generados en los diversos procesos.

Realizar el despliegue de los procesos implementados en las fiscalías de manera que mejore la gestión de información de los procesos fiscales.

Bibliografía

1. Giron, Sandra Milena Moran. *Incidencia de las TIC, Innovacion y Capital Humano.* Santiago de Cali : s.n., 2010.

2. Biblioteca Cepal. [En línea] [Citado el: 6 de febrero de 2014.] http://biblioguias.cepal.org/content.php?pid=411224&sid=3363230.

3. *Ley de la Fiscalía General de la República.* Habana : s.n., 1997.

4. Fiscalía General de la República. Instrucciones de Verificaciones Fiscales. Habana : s.n., 1999.

5. Figueroa, Yenier. *Proyecto Técnico SIGEFII.* UCI, La Habana : s.n.

6. República, Fiscalía General de la. *Explicación del negocio de las interfaces de Ejecución.* La Habana : s.n., 2013.

7. SONDA. [En línea] [Citado el: 22 de Enero de 2014.] www.sonda.com.

8. Sistema de Información del Ministerio Fiscal. [En línea] [Citado el: 24 de Enero de 2014.] http://www.astic.es/sites/default/files/articulosboletic/mono_6_1.pdf.

9. *InfoFiscalia.* Mulet, Marisa del Valle. Madrid : s.n., 2010.

10. Lex-Doctor Gestión Jurídica. [En línea] [Citado el: 2014 de febrero de 12.] http://www.lex-doctor.com/about.php..

11. [En línea] [Citado el: 26 de enero de 2014.] http://www.fiscalias.gob.ar/administracion/recursos/tecnologicos.

12. Fuentes Blancos , Hector, y otros. *Propuesta de arquitectura de software para proyectos de gestión sobre plataformas web.* UCI, La Habana : s.n.

13. Greddy Booch, Ivar Jacobson, James Rumbaugh. *El preceso unificado de desarrollo de software.* Madrid : Pearson Education , 2000.

14. González Obregon, William. *Modelo de desarrollo de software.* La Habana : s.n., 2012.

15. UML. [En línea] [Citado el: 6 de Diciembre de 2013.] http://www.eumed.net/libros-gratis/2009c/587/Lenguaje%20de%20Modelado%20Unificado.htm.

16. PHP. [En línea] [Citado el: 6 de Diciembre de 2013.] http://www.php.net/.

17. Desarrollo web. [En línea] [Citado el: 4 de Diciembre de 2013.] http://www.desarrolloweb.com/articuloscopyleft/articulo-metricas-de-software.html.

18. W3C. [En línea] [Citado el: 10 de Diciembre de 2013.] http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo..

19. Pacheco, Nacho. Manual de Twig. 2011.

20. Publispain. [En línea] [Citado el: 10 de Diciembre de 2013.] http://www.publispain.com/supertutoriales/Javascript/Intro.doc.

21. Visual Paradigm. [En línea] [Citado el: 12 de Diciembre de 2013.] http://www.visual-paradigm.com/.

22. PgAdmin3. [En línea] [Citado el: 12 de Diciembre de 2013.] http://www.pgadmin.org/.

23. Postgres, Comunidad. Comunidad Postgres. [En línea] [Citado el: 2 de diciembre de 2013.] http://postgresql.uci.cu.

24. PostgreSQL. [En línea] [Citado el: 11 de Diciembre de 2013.] http://www.postgresql.org.es/sobre_postgresql.

25. Netbeans. [En línea] [Citado el: 11 de Diciembre de 2013.] https://netbeans.org.

26. Symfony. [En línea] [Citado el: 12 de Diciembre de 2013.] http://symfony.com.

27. Doctrine. [En línea] [Citado el: 6 de Diciembre de 2013.] http://sf2-es.net16.net/doctrine/orm-documentation/reference/introduction.html.

28. Apache. [En línea] [Citado el: 5 de Diciembre de 2013.] http://httpd.apache.org/.

29. Subversion. [En línea] [Citado el: 20 de Enero de 2014.] http://subversion.apache.org.

30. Larman, Craig. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 2da Edición. Mexico : PRENTICE HALL, 1999. ISBN 970-17-0261-1.

31. Astudillo, Marcello Visconti y Hernán. *Fundamentos de Ingeniería de Software.* Mexico : Prentice Hall, 1999. ISBN 970-17-0261-1.

32. MSDN. [En línea] [Citado el: 9 de febrero de 2014.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.

33. [En línea] [Citado el: 8 de febrero de 2014.] http://www.albatronic.com/introduccion-al-paradigma-modelo-vista-controlado.

34. Muñoz, Dra. Coral. *Métricas de la Calidad de Software.* 2007 : s.n.

35. Soldado, Rosana Montes. *Métricas Aplicables al Diseño Orientado a Objetos.* Granada : s.n., 2011.

36. Gestión de Proyectos y Desarrollo de Software. [En línea] [Citado el: 2 de Marzo de 2014.] https://jummp.wordpress.com.

37. [En línea] [Citado el: 19 de febrero de 2014.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo6.pdf.

38. Pruebas de Software. [En línea] [Citado el: 10 de Febrero de 2014.] http://pruebasdesoftware.com/laspruebasdesoftware.html.

39. *Pruebas de caja blanca y caja negra.* UCI, La Habana : s.n., 2012.

40. Ingeniería de Software II. [En línea] [Citado el: 12 de Marzo de 2014.] http://sities.google.com/ingenieriadesoftwre.

41. Pressman, Roger S. *Ingeniería de software. Un enfoque práctico.*

42. Gutierrez, Demián. *UML Diagrama de Secuenia.* Venezuela : s.n., 2011.

43. [En línea] [Citado el: 8 de abril de 2014.] http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html.

44. República, Fiscalía General de la. *Constitución de la República.* La Habana : s.n.

45. Kruchten. *The Relational Unifeid Process: An Introduction.* s.l. : Adison Wesley, 2000.

46. Sierra, Francisco Gomez. Control Fiscal y REsponsabilidad Fiscal. [En línea] [Citado el: 24 de Diciembre de 2013.] http://edileyer.com.

47. Sommerville, Ian. *Ingeniería del software.* Septima Edición. s.l. : Pearson educación, 2005. ISBN: 84-7829-074-5.

48. Gómez, José Jorge Márquez. *Arquitectura MVC.Visión general.* 2009.

49. Usaloa, Macario Polo. *Patrones GRASP.* 2000.

50. Eguiluz, Javier. *Desarrollo Web Ágil con Symfony 2.* 2012.

51. SencioLabs. [En línea] [Citado el: 2014 de Febrero de 13.] http://twig.sensiolabs.org/.

52. Cuba, Fiscalía General República de. *Ley No 83, Artículo 8 ley de la Fiscalía General República.* Habana, Cuba : s.n., 1997.

53. Molpeceres, Alberto. *Procesos de desarrollo: RUP, XP y FDD.* 2003.