UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

Centro de Gobierno Electrónico

“Componente para la gestión de plantillas de documentos jurídicos en el Sistema de Gestión Fiscal”

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

Autor (es):
Erik Janser Silva Bueno

Tutor (es):
Ing. Yamila Ortuño Sánchez

Ing. Kenia López Hernández

Ing. José Carlos Arencibia Pérez

La Habana, junio de 2019
“Año 60 de la Revolución”

DERECHOS DE AUTOR

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Erik Janser Silva Bueno

Autor

_____ _____

Ing. Yamila Ortuño Sánchez **Ing. José Carlos Arencibia Pérez**

Tutor **Tutor**

_____

Ing. Kenia López Hernández

Tutor

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

Foto: Liborio Noval.

«(…) La informática se convertirá en una poderosísima fuerza científica, económica e incluso política del país (…)».

Fidel Castro

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

DATOS DE TUTOR

Datos de los tutores:

Nombre: Ing. Yamila Ortuño Sánchez

Correo electrónico: yamila@uci.cu

Nombre: Ing. Kenia López Hernández

Correo electrónico: klopez@uci.cu

Nombre: Ing. José Carlos Arencibia Pérez

Correo electrónico: jcarencibia@uci.cu

Datos del autor:

Nombre: Erik Janser Silva Bueno

Correo electrónico: ejsilva@estudiantes.uci.cu

Síntesis del tutor (es):

- Ing. Kenia López Hernández: Graduado de Ingeniero en Ciencias Informáticas en el año 2014 en la Universidad de las Ciencias Informáticas (UCI). Especialista B del Centro de Gobierno Electrónico. Se ha desempeñado como analista del Sistema de Informatización para la gestión de las fiscalías cubanas.
- Ing. Yamila Ortuño Sánchez: Graduado de Ingeniero en Ciencias Informáticas en el año 2016 en la Universidad de las Ciencias Informáticas (UCI). Especialista B del Centro de Gobierno Electrónico. Se ha desempeñado como analista y jefa de proyecto del Sistema de Informatización para la gestión de las fiscalías cubanas
- Ing. José Carlos Arencibia Pérez: Graduado de Ingeniero en Ciencias Informáticas en el año 2017 en la Universidad de las Ciencias Informáticas (UCI). Especialista B del Centro de Gobierno Electrónico. Se ha desempeñado como desarrollador del Sistema de Informatización para la gestión de las fiscalías cubanas.

DEDICATORIA

Antes que todo quiero decir que no soy bueno para estas cosas, aun haciéndolas ya estaba llorando, espero no llorar ahora. A la primera persona que le voy a agradecer es al Comandante en Jefe dado que un día se lo prometí y hoy lo estoy cumpliendo. También le agradezco a este claustro de profesores que me acompaño todos estos años para que llegara a este día no con todo el conocimiento porque yo nunca fui de estudiar, pero si con el necesario para poder decir hoy, soy ingeniero en ciencias informáticas. Agradezco mucho pero mucho a mi jefa de año quien fue la que más batallo con cada uno de nosotros, nos peleó, nos dijo de todo, pero en este momento ella también se merece un título por nunca dejarnos solo y no cansarse en el camino. Agradezco a mi jurado, agradezco a mi oponente, agradezco a cada uno de mis tutores quienes me guiaron en todo el transcurso de la tesis, pero también agradezco a una persona que, sin ser mi tutor ni tener que ver conmigo me ayudo en todo momento que lo necesite, por eso estoy orgulloso de haber sido la persona con más tutores en el mundo mundial, porque a pesar de no ser oficial y no haberlo nombrado en la presentación puedo decir que no tuve 3 sino 4 tutores. Ahora doy paso a mis compañeros a quienes les estoy agradecido por estar ahí a mi lado a lo largo de estos 5 años y espero seguir estando presente para cada uno de ustedes, también a mis hermanitos con los cuales llevo aún más tiempo recorrido y seguirán pasando los años y seguirán siendo mis hermanos. Otros que también son importantes para mi son esos amigos que conocí por casualidad pero que también los llevo en el corazón y se han convertido en hermanos míos aquí en esta religión que se llama universidad, a cada uno de ustedes mil gracias. Llegó el momento de mi familia, ahora si me toca llorar bastante, voy a empezar por ustedes 4 que me acompañan desde donde quiera que estén y me dan mucha luz y me cuidan siempre, para el que no entendió, los quiero abuelos. A ti mi carnalito, por más que nos fajemos recuerda siempre esto, un hermano es aquel amigo que el destino te regala para toda la vida, y tu eres el mejor de todos, te quiero mucho. A ti mi primo que sé que estas orgulloso de mi, ahora nos toca comernos el mundo. A ti papá que a tu manera y en tus posibilidades me has criado y me has dado todo lo que está a tu alcance, la vida no da la posibilidad de escoger los padres, pero de ser así te habría escogido a ti. Ahora te toca a ti mi amor, no voy hablar mucho de ti porque estamos en horario de almuerzo y las personas aquí ya tienen hambre, pero si voy a decirte que eres el centro de mi vida y que contigo quiero formar una familia linda donde los más importante sea la felicidad y el amor, te amo con mi vida. Y por último a la mujer más importante en mi vida, la mujer que me dio este tamaño, este carácter, y que se robó todo el amor del mundo para dárselo a su hijo el cual está más que agradecido por ello, mamita somos ingenieros en ciencias informáticas. Al resto de personas que tuvieron que ver de una forma u otra, mil gracias.

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

RESUMEN

La Fiscalía General de la República de Cuba, es la institución rectora del sistema judicial cubano y el órgano del estado, responsable del control y conservación de la legalidad dentro del sistema social. Como parte del esfuerzo del gobierno cubano de informatizar paulatinamente todos los sectores de la sociedad, se beneficia el sector jurídico. Actualmente se encuentra en funcionamiento en todas las fiscalías del país, un software para la informatización de procesos fiscales denominado Sistema de Gestión Fiscal, el cual fue desarrollado por la Universidad de las Ciencias Informáticas, para agilizar el trabajo de los fiscales y elevar los niveles de conformidad por parte de los ciudadanos cubanos.

El presente trabajo abarca la implementación del componente para la gestión de las plantillas de documentos jurídicos en el Sistema de Gestión Fiscal, el componente permite a los fiscales editar las plantillas de documentos jurídicos de una manera más sencilla y eficiente, las modificaciones son a nivel de edición de texto sin necesidad de un especialista informático como pasaba antes. Este, es un paso importante para mejorar la interacción de los fiscales con el software, así como contribuir a la usabilidad del mismo.

Como fundamento de la investigación, se realiza un estudio de los sistemas similares y se definen lenguajes y tecnologías para el desarrollo de la solución propuesta. A partir del análisis y las entrevistas con el cliente se obtienen los requisitos del sistema. Además, se aplican métricas y pruebas de software para garantizar la calidad del producto.

Palabras claves:

Gestión, editar, plantillas, documentos, sistema

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

ABSTRACT

The Attorney General of the Republic of Cuba, is the governing institution of the Cuban judicial system and the state body, responsible for the control and conservation of legality within the social system. As part of the Cuban government's effort to gradually computerize all sectors of society, the legal sector benefits. Currently, a software for the computerization of tax processes called Fiscal Management System, which was developed by the University of Computer Sciences, is in operation in all the country's prosecutor's offices, to speed up the work of prosecutors and raise the levels of compliance on the part of Cuban citizens.

The present work covers the implementation of the component for the management of the legal document templates in the Fiscal Management System, the component allows the prosecutors to edit the templates of legal documents in a more simple and efficient way, the modifications are at the level of text editing without the need of a computer specialist as happened before. This is an important step to improve the interaction of prosecutors with the software, as well as contribute to its usability.

As a basis for the research, a study of similar systems is carried out and languages and technologies are defined for the development of the proposed solution. From the analysis and interviews with the client, the system requirements are obtained. In addition, metrics and software tests are applied to guarantee the quality of the product.

***Keywords**:*

Management, edit, templates, documents, system

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones

Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.

Teléfono: +53 7 8358025 y +53 7 8373821

ÍNDICE

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

ÍNDICE DE TABLAS

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

ÍNDICE DE FIGURAS

Universidad de las Ciencias Informáticas. Centro de Gobierno Electrónico. Departamento Desarrollo de Aplicaciones
Carretera San Antonio Km 2 ½ Torrens. Boyero. La Habana. Cuba.
Teléfono: +53 7 8358025 y +53 7 8373821

INTRODUCCIÓN

Las Tecnologías de Información y Comunicación (TICs) son el conjunto de herramientas relacionadas con la transmisión, procesamiento y almacenamiento digitalizado de la información. Un aliado del emprendimiento, tanto en nuevos conceptos como en lo tradicional. Las nuevas generaciones están más que acostumbradas a conceptos como community mannager, marketing digital, redes sociales, tiendas online o transformación digital, conceptos que hace diez años apenas comenzaban a vislumbrarse. Estos nuevos modelos han sido consecuencia de las nuevas TICs, que han revolucionado el mercado laboral. De acuerdo con la Universidad Nacional Autónoma de México (UNAM), las TICs contemplan al conjunto de herramientas relacionadas con la transmisión, procesamiento y almacenamiento digitalizado de la información, como al conjunto de procesos y productos derivados de las nuevas herramientas (hardware y software), en su utilización en la enseñanza (Naser A. 2011).

Entre las instituciones que están inmersa en el proceso de utilización de las TICs se encuentra la Fiscalía General de la República (FGR). Esta institución es el Órgano del Estado al que corresponde, como objetivos fundamentales, según el artículo 127, capítulo XIII, de la Constitución de la República de Cuba, 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. De igual forma la promoción y el ejercicio de la acción penal pública en representación del Estado. Su misión institucional es proteger el orden político y jurídico del Estado y la sociedad, procurando el restablecimiento de la legalidad quebrantada promoviendo acciones y medidas contra los infractores (Cuervo J. 2015).

La Universidad de las Ciencias Informáticas (UCI), fundada en el año 2002, es una institución adscrita al Ministerio de Educación Superior de la República de Cuba. Es un centro docente-productor que forma profesionales altamente comprometidos y calificados en la rama de la Informática, mediante un modelo de formación atípico y flexible basado en la vinculación estudio –trabajo. La actividad de desarrollo de software está evaluada con la categoría CMMI Nivel 2 (Martínez M. 2015).

Como parte de un acuerdo entre la FGR y la UCI; se crea el Sistema de Gestión Fiscal (SIGEF) para agilizar el trabajo de los fiscales. Este sistema fue desarrollado en la universidad, específicamente en el Centro de Gobierno Electrónico (CEGEL) de la Facultad 3. SIGEF es una aplicación web que tiene como objetivo principal informatizar los procesos que se desarrollan en las fiscalías, para agilizar la toma de decisiones y elevar el control de los procesos.

INTRODUCCIÓN

La forma en que las entidades y órganos del estado administran y ejecutan sus procesos, cambian radicalmente mediante el uso de las TIC, constituyendo un medio para que los gobiernos modernicen sus procesos, mejoren sus interacciones con los ciudadanos y disminuyan la brecha digital existente entre estos. El SIGEF está formado por 15 módulos que informatizan los procesos jurídicos que desarrolla la Fiscalía General de la República de Cuba (FGR), se ha implantado con buenos resultados en todas las instancias de la FGR a nivel de país (169 municipios, 15 provincias) con más de 1800 usuarios que gestionan los procesos asignados a cada territorio garantizando con ello eficiencia y eficacia de la gestión fiscal, brindándole a los ciudadanos un marco de transparencia, celeridad, imparcialidad, honestidad y buena fe teniendo en cuenta las disposiciones legales que regula la Constitución de la República de Cuba (Cuervo J. 2015).

Descripción de la necesidad y la situación problemática de la investigación

La implementación y despliegue del SIGEF constituye un resultado de alto impacto para la entidad. Los cambios en la forma de manejar y gestionar los procesos por los trabajadores es uno de los principales retos, donde adaptarse al sistema es el eje fundamental. La solución, es la primera de su tipo puesta en funcionamiento en las fiscalías del país. Entre los principales beneficios que ofrece están ahorrar los materiales de uso frecuente, acortar el tiempo de tramitación y ofrecer un conjunto de estadísticas fiables de todos los documentos procesados, y así favorecer con claridad la toma de decisiones.

Con el tiempo de uso del SIGEF, surgen inquietudes por parte de la fiscalía. La captación de los datos primarios de cada documento posibilita a que estén expuestos a la introducción de errores. Los escritos jurídicos, sobre todo los que responden a trámites y procesos judiciales, son individualizados. Un asunto de similar e incluso, idéntica naturaleza y contenido, se diferencian en dependencia de las circunstancias que identifican a cada caso. Ese es el caso de las demandas, los escritos de contestación, los dictámenes y otros documentos que se emiten por la Fiscalía al intervenir en cada uno. De ahí que, aunque se pueda admitir alguna presentación general, común para muchos, debe existir la posibilidad de ajustarlos a los requerimientos y características de cada asunto.

En la práctica un escrito puede ser elaborado y revisado en varias ocasiones para completarlo, ampliarlo y hasta mejorar su redacción. Esto sucede hasta después de concluido y en las condiciones actuales eso no lo permite el sistema, lo cual obliga a conformarse con un documento que a juicio de los que lo analizan no es totalmente correcto. En esa circunstancia hay que apelar a la ampliación u otra vía sin necesidad, pues hasta tanto no se presente el documento a la institución correspondiente puede ser susceptible de modificación.

Los documentos de los módulos del SIGEF tienen un pie de firma con los datos del jefe en cuestión y en el encabezado se muestran los datos formales de la persona a la cual se le va a enviar dicha

información. Estos datos se mantienen fijos y no se pueden modificar además de las formalidades que los fiscales utilizan y que usualmente usan en todos sus documentos.

También surgen cuestiones de trabajo que involucran a los jefes en estas tareas y no están presentes en el órgano central, lo cual provoca que la persona que firma no sea la que está plasmada en la plantilla o documento y la responsabilidad se le otorga a otra persona que lo sustituya lo cual no es variable actualmente en el SIGEF. El cuerpo de documento varía también en casi todos los casos porque la realidad es que todos no son iguales y es necesario informar algunas cuestiones a los involucrados en el proceso que no son posibles como están diseñadas las plantillas.

Para modificar la plantilla es necesario la ayuda de un especialista en el desarrollo que pueda asistir a la instancia de la Fiscalía General y resolver los cambios trabajando a nivel de código lo cual requiere además conocimientos técnicos para hacerlo. Esto provoca que los fiscales usen el SIGEF por formalidad en casi todos los casos**,** pues al final de la introducción de los datos que le son necesarios para trabajar, los resultados son los documentos que dichos datos generan, que al no corresponderse con la información que desean almacenar, se ven obligados a realizar esta plantilla nuevamente en algún editor de texto. Todo lo antes mencionado trae consigo lentitud en el proceso y conduce a que se invierta tiempo, recursos humanos y materiales. De acuerdo a lo antes expuesto, se evidencia la necesidad de editar las plantillas, ya que en estos momentos no se permite a los fiscales gestionar las plantillas de los procesos de forma personalizada e individual.

De acuerdo con la situación problemática expuesta anteriormente se identifica como **problema de investigación:** ¿Cómo gestionar las plantillas de documentos jurídicos en el SIGEF, de forma tal que se garantice la usabilidad en la conformación de los documentos en dicho sistema?

Por lo que se define como **objeto de estudio**: proceso de desarrollo de software en la Informática Jurídica de Gestión, enmarcándose en el **campo de acción**: desarrollo de un componente para la gestión de plantillas de documentos jurídicos en el SIGEF.

En función de dar respuesta al problema identificado se traza como **objetivo general:** desarrollar un componente para la gestión de plantillas de documentos jurídicos en el SIGEF, de forma tal que garantice la usabilidad en la conformación de los documentos en dicho sistema.

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

- Identificar los referentes teóricos en los que se sustenta la propuesta de solución para facilitar el entendimiento del objeto de estudio de la presente investigación.

- Realizar el análisis y diseño de la propuesta de solución como aproximación a la implementación.
- Implementar el componente para la gestión de plantillas de documentos jurídicos en el SIGEF para garantizar la usabilidad en la conformación de los documentos en dicho sistema.
- Validar la solución propuesta mediante pruebas de software para avalar la calidad del componente propuesto.

Se plantea como **idea a defender:** si se desarrolla un componente de software que permita la gestión de las plantillas de documentos jurídicos en el SIGEF, se contribuye a la usabilidad del sistema.

Proponiendo como **posibles resultados:** una vez finalizado el presente trabajo, se tendrá como resultado el componente para garantizar la usabilidad en la conformación de los documentos en el SIGEF.

Para realizar las tareas de investigación se ponen en práctica los siguientes Métodos científicos:

Métodos teóricos:

- Analítico-sintético: se pretenden integrar los conocimientos del proceso de desarrollo del software para su aplicación en la informática jurídica, se analizan los conceptos de gestión de plantillas de procesos para un mayor acercamiento a los elementos que engloban la presente investigación. Además de realizar un procesamiento de la información para llegar a conclusiones prácticas y teóricas sobre la investigación.
- Modelación: permite crear abstracciones con vistas a explicar la realidad y utilizar el modelado como sustituto del objeto de la investigación. Además, es empleado para modelar los procesos de negocio relacionados con la fiscalía.

Métodos empíricos:

- Observación**:** con este método es posible recopilar una serie de datos confiables, que ayudan a la comprensión del fenómeno y a la definición 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 lograrlo.
- Entrevista: al entrevistar al cliente, la utilización de este método permitió definir previamente el trabajo, dejando documentado los requisitos funcionales y la especificación de cada uno de ellos.

- Tormenta de ideas: permitió la recopilación de un conjunto de datos relacionados con el sistema, lo cual proporcionó un mayor entendimiento de la lógica de negocio del SIGEF y de los módulos especificados.

El presente trabajo está estructurado en tres capítulos. A continuación, se presenta una breve descripción de cada uno de ellos.

En el **Capítulo 1 "Fundamentación teórica"** se describen los conceptos asociados al dominio del problema, se realiza un estudio del estado del arte haciendo un análisis de las principales tecnologías de desarrollo de software en la actualidad, del cual se obtienen las principales opciones en cuanto a técnicas de programación, herramientas, marcos de trabajo y gestores de base de datos a utilizar. También se estudian las métricas para evaluar el diseño de software, los patrones arquitectónicos y de diseño; además de las distintas pruebas que se realizan al software para validar la calidad de la solución y detectar errores.

En el **Capítulo 2 "Descripción de la propuesta"** se expone el modelado del negocio y la descripción de los procesos de negocio. Se definen los requisitos tanto funcionales como no funcionales, teniendo en cuenta las restricciones o políticas a cumplir por el sistema. Se aplican los patrones arquitectónicos y de diseño, se definen los estándares de codificación utilizados.

En el **Capítulo 3 "Validación de la propuesta"** a partir de los resultados del capítulo anterior se realiza la validación de la solución propuesta a través de las pruebas internas, de liberación y de aceptación con el fin de eliminar las no conformidades del sistema y así mejorar la calidad del mismo. Además, se aplican métricas para validar el diseño y se validan las variables de la investigación.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se abordan los elementos desde el punto de vista teórico que se tienen en cuenta para la propuesta de solución. Se estudian las principales características de varias aplicaciones jurídicas desarrolladas en el mundo con el fin de obtener los elementos significativos para la presente investigación. Se definen los patrones arquitectónicos y de diseño a utilizar, las métricas para evaluar el diseño de software y las distintas pruebas para validar los resultados. Además, se identifican y describen las herramientas y lenguajes a emplear en el presente trabajo.

1.2. Marco teórico

El marco teórico consiste en la recopilación de antecedentes, investigaciones previas y consideraciones teóricas por donde se sustenta un proyecto de investigación, análisis, hipótesis o experimento, permitiendo la interpretación de los resultados y la formulación de conclusiones. El marco teórico, también llamado como marco de referencia, es el soporte conceptual de una teoría o de los conceptos teóricos que se utilizaron para el planteamiento del problema de un proyecto o una tesis de investigación, por ejemplo. El marco teórico se caracteriza por tener un lenguaje teórico donde se define la disciplina a la cual pertenece el campo de estudio escogido, el o los conceptos relevantes y el fenómeno que se quiere profundizar o estudiar. La importancia del marco teórico radica en que permite, de forma ordenada y coherente, justificar, demostrar, apoyar e interpretar las hipótesis y los resultados de una investigación y, a su vez, formular de una forma confiable las conclusiones de un proyecto o replantear preguntas de niveles superiores de abstracción y profundidad (Significados.De 2018).

A continuación, la presentación de los principales conceptos que se abordan en la presente investigación.

1.2.1. Gobierno electrónico

“El Gobierno electrónico es una innovación continua de los servicios, la participación de los ciudadanos y la forma de gobernar mediante la transformación de las relaciones externas e internas a través de la tecnología, el Internet y los nuevos medios de comunicación” (Naser y Concha 2011).

“El Gobierno Electrónico es el uso de las tecnologías de la información y comunicación (TICs), particularmente la Internet, como una herramienta para alcanzar un mejor gobierno” (OCDE)

Gobierno Electrónico = Gobierno + TICs + Servicios + Sociedad

En la política integral para el perfeccionamiento de la informatización de la sociedad cubana, aprobada en febrero de 2017 y publicada en el sitio del Ministerio de Comunicaciones, se define como Gobierno Electrónico:

“El uso de las TIC en los órganos de la Administración para mejorar la información y los servicios ofrecidos a los ciudadanos, orientar la eficacia y eficiencia de la gestión pública e incrementar sustantivamente la transparencia del sector público y la participación de los ciudadanos”. (Concha G. 2011)

1.2.2. Informática jurídica de gestión

La Informática jurídica de gestión es conjunto de aplicaciones de la informática en el ámbito jurídico; es una técnica interdisciplinaria que tiene por objeto el estudio e investigación de los conocimientos aplicables a la recuperación de información jurídica, así como la elaboración y aprovechamiento de los instrumentos de análisis y tratamiento de información jurídica, necesaria para lograr dicha recuperación.

La informática jurídica pretende dar una base de información a los investigadores en el campo del Derecho Informático sobre Legislación, Sentencias Judiciales, Master, Seminarios, Congresos, Cursos, Directorios Jurídicos de Internet y Bibliografía (Cubadebate 2018).

1.2.3. Usabilidad

La usabilidad web es el grado de facilidad de uso que tiene una página web para los visitantes que entran e interactúan con ella. Una web con una buena usabilidad es aquella que permite a los usuarios una interacción sencilla, intuitiva, agradable y segura. Para lograr mejorar la usabilidad de una web no se debe perder de vista la siguiente idea u objetivo principal: poner todos los medios para que los visitantes puedan encontrar lo que buscan con el mínimo esfuerzo y en el menor tiempo posible (Egea C. 2007).

1.2.4. Técnicas de programación

Programación estructurada

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while); asimismo, se considera innecesario y contraproducente el uso de la instrucción de transferencia incondicional (GOTO), que podría conducir a código espagueti, mucho más difícil de seguir y de mantener, y fuente de numerosos errores de programación. (García-Bermejo Giner, José Rafael 2008)

Con posterioridad a la programación estructurada se han creado nuevos paradigmas tales como la programación modular, la programación orientada a objetos, la programación por capas y otras, así como nuevos entornos de programación que facilitan la programación de grandes aplicaciones y sistemas. (García-Bermejo Giner, José Rafael 2008)

Programación Orientada a Objetos (POO)

La POO es una técnica para desarrollar soluciones computacionales utilizando componentes de *software* (objetos de *software*).

Objeto: Componente o código de *software* que contiene en sí mismo tanto sus características (campos) como sus comportamientos (métodos); se accede a través de su interfaz o signatura.

Campo: Es una característica de un objeto, que ayuda a definir su estructura y permite diferenciarlo de otros objetos. Se define con un identificador y un tipo, el cual indica los valores que puede almacenar. El conjunto de valores de los campos define el estado del objeto.

Método: Es la implementación de un algoritmo que representa una operación o función que un objeto realiza. El conjunto de los métodos de un objeto determina el comportamiento del objeto.

La POO es un paradigma de la programación de computadores; esto hace referencia al conjunto de teorías, estándares, modelos y métodos que permiten organizar el conocimiento, proporcionando un medio bien definido para visualizar el dominio del problema e implementar en un lenguaje de programación la solución a ese problema. La POO se basa en el modelo objeto donde el elemento principal es el objeto, el cual es una unidad que contiene todas sus características y comportamientos en sí misma, lo cual lo hace como un todo independiente pero que se interrelaciona con objetos de su misma clase o de otra clase, como sucede en el mundo real.

Anterior al paradigma de objetos, está el paradigma algorítmico o de procesos, el cual se fundamenta en los procesos o funciones que se llevan a cabo en el mundo real dentro del dominio del problema analizado. Se refiere a lo que entra, como lo maneja el proceso, y lo que sale del proceso. La programación tradicional la sustentan los procesos, algoritmos, bloques de construcción modulares cuya abstracción va de lo general a lo particular, mientras que en la POO tiene como marco de referencia conceptual el objeto, el cual pertenece a una clase que agrupa a todos compañeros con las mismas características y un comportamiento similar.

Existe un acuerdo acerca de qué características contempla la "orientación a objetos", las características siguientes son las más importantes:

- Abstracción
- Encapsulamiento
- Principio de ocultación
- Polimorfismo
- Herencia

Luego de ser analizadas las dos técnicas de programación anterior se escoge para la implementación del software la POO debido que esta técnica permite la reutilización del código para facilitar el trabajo en conjunto y permite la creación de sistemas complejos agilizando el desarrollo del software.

1.3. Estado del arte

En las diversas fiscalías alrededor del mundo se generan volúmenes grandes de información sobre los casos manejados por los diferentes fiscales, estos volúmenes son generados por los engorrosos procedimientos llevados a cabo, por tanto, el uso eficiente de las TICs ha hecho posible crear sistemas fiscales informatizados para lograr un mayor flujo de la información y transparencia en los procesos. La FGR actualmente cuenta con el SIGEF en su segunda versión, pero este no cuenta con un gestor de plantillas que ayude a gestionar cada uno de los procesos fiscales, por tanto, es necesario hacer un estudio de los sistemas de informática jurídica existentes en el mundo, con el objetivo de analizar si cuentan con un componente capaz de darle solución al problema presentado, de esta manera se identificaran tendencias y adoptara una posición al respecto.

Soluciones internacionales existentes

CaiFis: es un software de fiscalidad para la gestión e impresión de documentos necesarios en el departamento fiscal, con un precio económico, y versiones que varían en funcionalidades y/o volumen de datos, para adaptarse a cualquier negocio (asesorías, grandes empresas, autónomos, pymes, etc.), entre las ventajas que presenta están:

- Trabaja ágilmente manejando opciones de programa concretas para cada proceso fiscal.
- Configura las opciones del programa y obtendrás la información fiscal según las necesidades de tu empresa.
- Introduce datos e información fiscal una sola vez y sácale partido en diferentes cálculos de impuestos.
- Benefíciate de la apariencia y diseño profesional de toda la documentación generada por el programa.

SIPPAU: es la solución informática desarrollada por la Fiscalía General de la Nación en Uruguay. A partir del miércoles 6 de setiembre de 2017, en las sedes de la Fiscalía General de la Nación del interior del país, comenzó la implementación del módulo no acusatorio del Sistema de Información de la Fiscalía llamado SIPPAU, permitiendo a esas fiscalías la gestión digitalizada de las actuaciones en todas las materias. El módulo no acusatorio comprende los expedientes que se tramitan actualmente, no aquellos que comenzarán con el nuevo proceso penal lo que será producto de una etapa más avanzada del sistema en la que se está trabajando. El software permite el registro digital de las actuaciones, así como la gestión de los diferentes documentos. Entre otras ventajas, esta herramienta facilita el acceso a la información y da mayores garantías a los más de 230 funcionarios que trabajan en las 33 sedes del interior del país.

Lex-Doctor: es un conjunto de softwares jurídicos desarrollado en Argentina por Sistemas Jurídicos. Su tecnología de administración de datos, permite manejar grandes volúmenes de información, y organizar grupos de trabajo operando en redes de gran cantidad de terminales; todo ello, con los más bajos costos de instalación, gracias a que el licenciamiento no requiere el pago adicional de licencias de procesadores de texto o motores de datos, los cuales son generalmente mucho más costosos que la aplicación en sí misma.

Gedex: es un software para las fiscalías, pero solo para plataformas Windows. GEDEX realiza el seguimiento completo de los expedientes jurídicos de su despacho, bufete o departamento jurídico. Es el sistema más seguro, y el más rápido para buscar y localizar sus casos, con diferencia. GEDEX le ayudará a incrementar beneficios y ahorrar en costes de gestión, de la forma más natural. Informatizar sus expedientes no implica en absoluto cambiar su forma de trabajo habitual y la tradición de su firma, ni destruir el papel o usar facturas electrónicas.

Solución existente en Cuba

SIGEF en su versión 2 está dirigido a adicionar nuevas funcionalidades al SIGEF en su primera versión anteriormente implementado, el uso de las nuevas tecnologías dio lugar al SIGEF II, en este sentido como valor agregado se tiene la obtención de nuevos componentes para una mayor utilidad del sistema, SIGEF II usa plataforma libre y es el encargado de gestionar todos los procesos en las fiscalías cubanas. (Figueroa, 2011)

A continuación, se comparan los diversos sistemas analizados con respecto a las variables Usabilidad, Plataformas Libres y Software libre de costos.

Nombres	Usabilidad	Plataformas libres	Software libre de costos
CaiFis	**X**		
SIPPAU	**X**		
Lex-Doctor	**X**		
Gedex	**X**		
SIGEF2		**X**	**X**

Fuente: elaboración propia

Después de analizar la tabla anterior, se concluye que las soluciones internacionales difieren del esquema socialista establecido en Cuba, los costos elevados de estos softwares y además del mantenimiento de sus licencias de uso, dificultan su uso en el sistema de gestión fiscal cubano, debido

que no se puede acceder a ellos por la económica actual del país. Además, estos no presentan gestores de plantillas.

Los sistemas utilizados en Cuba por la FGR (SIGEF I y actualmente SIGEF II) por ser creados propiamente para satisfacer las necesidades de dicha institución, se adaptan a los procesos y procedimientos naturales del país y del sistema fiscal, pero estas soluciones carecen de un componente que permita gestionar las diversas plantillas referentes a cada proceso llevado a cabo por parte de los fiscales, solo cuentan con una plantilla base para gestionar todos los procesos, por lo que se hace necesario llevar a cabo la investigación en curso para dar solución al problema en cuestión.

1.4. Metodología de desarrollo

Una metodología de desarrollo de software, consiste principalmente en hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo. Regularmente este tipo de metodología, tienen la necesidad de venir documentadas, para que los programadores que estarán dentro de la planeación del proyecto, comprendan perfectamente la metodología y en algunos casos el ciclo de vida del software que se pretende seguir (Pressman 2010).

Se escogió como metodología a utilizar una variación de la metodología “Proceso Unificado Ágil” (AUP por sus siglas en inglés) en unión con el modelo CMMI-DEV v 1.3, desarrollada en la UCI, denominada AUP-UCI, la cual es idónea para la entrega de productos en cortos períodos de tiempo; además de satisfacer las necesidades del proceso (Ambler 2005).

De las disciplinas que propone AUP-UCI (Sánchez 2015) serán empleadas:

- Modelado de negocio

  Para modelar el negocio se proponen las siguientes variantes:

  1. Casos de Uso del Negocio (CUN).
  2. Descripción de Proceso de Negocio (DPN).
  3. Modelo Conceptual (MC).
- Requisitos: esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. Existen tres formas de encapsular los requisitos [Casos de Uso del Sistema (CUS), Historias de usuario (HU) y Descripción de requisitos por proceso (DRP)], agrupados en cuatro escenarios condicionados por el Modelado de negocio.

- Análisis y diseño: en esta disciplina se modela el sistema y su forma para que soporte todos los requisitos, incluyendo los requisitos no funcionales.
- Implementación: en la implementación, a partir de los resultados del análisis y diseño se construye el sistema.
- Pruebas internas: en esta disciplina se verifica el resultado de la implementación probando cada construcción, tanto internas como intermedias, así como las versiones finales a ser liberadas.
- Pruebas de liberación: pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.

Para la disciplina de requisitos, AUP en su variación UCI propone los siguientes escenarios:

Escenario No 1:

- Proyectos que modelen el negocio con Casos de Uso del Negocio (CUN) solo pueden modelar el sistema con Casos de Uso del Sistema (CUS).

Figura 14 Escenario 1 de AUP-UCI
Fuente:(Sánchez 2015)

Escenario No 2:

- Proyectos que modelen el negocio con Modelo Conceptual (MC) solo pueden modelar el sistema con Casos de Uso del Sistema (CUS).

Figura 15 Escenario 2 de AUP-UCI

Fuente:(Sánchez 2015)

Escenario No 3:

- Proyectos que modelen el negocio con Descripción de Proceso de Negocio (DPN) solo pueden modelar el sistema con Descripción de Requisitos por Proceso (DRP).

- Figura 16 Escenario 3 de AUP-UCI

Fuente:(Sánchez 2015)

Escenario No 4:

- Proyectos que no modelen negocio solo pueden modelar el sistema con Historias de Usuario (HU).

Figura 17 Escenario 4 de AUP-UCI

Fuente:(Sánchez 2015)

Para esta investigación, se escogió el escenario 3, ya que es el que mejor se adapta a las condiciones de desarrollo del sistema, ya que en la fiscalía existe un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad.

1.5. Descripción de lenguajes y herramientas a utilizar

A continuación, se definen los lenguajes y herramientas escogidos por parte del equipo de arquitectura del SIGEF.

1.5.1. Lenguajes

En el presente epígrafe se abordarán los lenguajes de modelado y de programación utilizados por el equipo de desarrollo en la realización del software SIGEF y de cada componente.

Lenguaje de Modelado Unificado (UML v2.0)

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el Object Management Group (OMG).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.
UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que programación estructurada es una forma de programar como lo es la orientación a objetos, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML solo para lenguajes orientados a objetos. (Fowler 2004).

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 (V 7.2)

PHP, acrónimo recursivo en inglés de PHP Personal Hypertext processor (preprocesador de hipertexto), es un lenguaje de programación de propósito general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en un documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera el HTML resultante.

PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en muchos sistemas operativos y plataformas sin ningún costo. Es también el módulo Apache más popular entre las computadoras que utilizan Apache como servidor web (PHP Group 2017)

Se utiliza en el desarrollo de la aplicación ya que brinda múltiples ventajas, y dado que Symfony es un framework para desarrollar en PHP.

HTML5

HTML5 (Hypertext Markup Language, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web, especifica dos variantes de sintaxis para HTML: una «clásica», HTML (text/html), conocida como HTML5, y una variante XHTML conocida como sintaxis XHTML5 que deberá servirse

con sintaxis XML (application/xhtml+xml). Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014. (Hickson, Hyatt 2009)

Se emplea porque el código es más simple que versiones anteriores, lo que permite hacer páginas más ligeras que se cargan más rápidamente favoreciendo la usabilidad.

CSS3

CSS (siglas en inglés de Cascading Style Sheets), en español "Hojas de estilo en cascada", es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML; el lenguaje puede ser aplicado a cualquier documento XML, incluyendo XHTML, SVG, XUL, RSS, etcétera. También permite aplicar estilos no visuales, como las hojas de estilo auditivas. Junto con HTML y JavaScript, CSS es una tecnología usada por muchos sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web, y GUIs para muchas aplicaciones móviles como Firefox OS. (World Wide Web Consortium 2016)

CSS está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o layouts, los colores y las fuentes. Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características presentacionales, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, y reducir la complejidad y la repetición de código en la estructura del documento. (World Wide Web Consortium 2016)

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

Twig

Twig es un motor de plantilla para el lenguaje de programación PHP que nace con el objetivo de facilitar el trabajo con las vistas, a los desarrolladores de aplicaciones web que utilizan la arquitectura Modelo-Vista-Controlador (MVC). Su sintaxis origina de Jinja y las plantillas Django. Es un producto de código abierto autorizado bajo Licencia BSD y mantenido por Fabien Potencier. La versión inicial estuvo

creada por Armin Ronacher. El framework Symfony3 para PHP viene con un soporte incluido para Twig como su motor de plantilla por defecto. (Potencier 2009)

Twig define tres clases de delimitadores:

1. {%... %}, se utiliza para ejecutar declaraciones, como pueden ser los bucles for.
2. {{...}}, se utiliza para imprimir el contenido de variables o el resultado de evaluar una expresión.
3. {#... #}, se utiliza para añadir comentarios en las plantillas. Estos comentarios no son incluidos en la página renderizada.

Se selecciona Twig debido a que con este motor las plantillas son más fáciles de hacer y resultan muy intuitivas. También implementa un novedoso mecanismo de herencia de plantillas, por lo que preocuparse por el peso que conlleva el interpretar todo ello no es un problema, debido a que guarda en clases PHP todo el contenido de las mismas, para acelerar el rendimiento de la aplicación.

JavaScript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo, en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Se escogió *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.

1.5.2. Herramientas

A continuación, se mencionan las diferentes características de las herramientas utilizadas para el desarrollo del componente en cuestión.

Visual Paradigm for UML v15.1

Visual Paradigm es una herramienta CASE: (Ingeniería de Software Asistida por Computación). La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. (Sierra 2017)

La herramienta está diseñada para un amplio rango de usuarios, incluyendo ingenieros de software, analistas de sistema, analistas de negocio, arquitectos de sistemas y cualquiera que esté interesado en construir un sistema confiable a gran escala a través del uso del enfoque orientado a objetos. La transición del análisis al diseño y luego a la implementación se realiza sin percances con este tipo de herramienta CASE, por lo que disminuye significantemente el esfuerzo empleado en todas las etapas del ciclo de vida del desarrollo. (Sierra 2017)

Se decide escoger el Visual Paradigm for UML pues 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 escogido. Además, la universidad posee una licencia para su uso.

PgAdmin 3

Entorno de escritorio visual libre y de código abierto. Instalable en plataformas Linux, FreeBSD, Solaris, Mac OSX y Windows. Permite conectarse a bases de datos PostgreSQL que estén ejecutándose en cualquier plataforma. Está disponible en diferentes idiomas. Facilita la gestión y administración de bases de datos ya sea mediante instrucciones SQL o con ayuda de un entorno gráfico. Permite acceder a todas las funcionalidades de la base de datos; consulta, manipulación y gestión de datos, incluso opciones avanzadas como manipulación del motor de replicación Slony-I. (Lelarge 2014).

En diciembre de 2014 Dave Page, anunció que, con el cambio hacia modelos basados en web, se comenzó a trabajar en pgAdmin 4 con el objetivo de facilitar los despliegues en la nube. PgAdmin 4 fue presentado el 29 de septiembre de 2016 fue hecho en Python y Javascript/jQuery, y un runtime para escritorio escrito en C ++ con la librería gráfica Qt.

Es escogido pgAdmin 3 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.

PostgreSQL v10.1

PostgreSQL es un sistema de gestión de bases de datos relacionales orientado a objetos y de código abierto, publicado bajo la licencia PostgreSQL. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y apoyados por organizaciones comerciales. PostgreSQL no tiene un gestor de defectos, haciendo muy difícil conocer el estado de sus defectos.

Entre las ventajas que presenta se puede encontrar:

- Seguridad en términos generales
- Integridad en BD: restricciones en el dominio

Incorpora funciones de diversa índole como manejo de fechas, geométricas y orientadas a operaciones con redes. Permite la declaración de funciones propias, así como la definición de disparadores. Soporta el uso de índices, reglas y vistas. Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos. (Bouchareine 2014).

Es seleccionado PostgreSQL como gestor de base de datos para el desarrollo del trabajo debido que brinda seguridad en el trabajo con base de datos, además de gozar de un gran prestigio en la comunidad internacional.

Doctrine v2.4

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 sistema de gestión de bases de datos. (Wage 2016)

Una característica de Doctrine es el bajo nivel de configuración que necesita para empezar un proyecto. Doctrine puede generar clases a partir de una base de datos existente y después el programador puede especificar relaciones y añadir funcionalidad extra a las clases autogeneradas. No es necesario generar o mantener complejos esquemas XML de base de datos como en otros frameworks. (Wage 2016)

Otra característica importante de Doctrine es la posibilidad de escribir consultas de base de datos utilizando un dialecto de SQL denominado DQL (Doctrine Query Language) que está inspirado en Hibernate (Java). (Wage 2016)

Se utiliza Doctrine debido a que tiene funciones de mapeo y consulta de objetos extremadamente flexibles y potentes. Además, viene integrado con el marco de trabajo seleccionado *Symfony3*.

IDE Netbeans v8.2

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso. (M. Domínguez-Dorado 2005)

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. (M. Domínguez-Dorado 2005)

Se utiliza NetBeans en su versión 8.2 para el desarrollo del componente, ya que permite la creación y depuración de proyectos PHP, además que provee integración con el SGBD PostgreSQL y con el *framework Symfony 3*. Además de estar el equipo de desarrollo familiarizado con dicha herramienta.

Marco de trabajo. Symfony v3.4

Symfony es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Este separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (Armand 2014).

Symfony fue diseñado para ajustarse a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y Unix-like estándares).

- Independiente del sistema gestor de bases de datos. Su capa de abstracción y el uso de ORM (Doctrine 2, Propel), permiten cambiar con facilidad de SGBD en cualquier fase del proyecto.
- Utiliza programación orientada a objetos y características como los espacios de nombres, de ahí que sea imprescindible PHP 5.3.* o superior.
- Sencillo de usar en la mayoría de casos, aunque es preferible para el desarrollo de grandes aplicaciones Web que para pequeños proyectos.
- Aunque utiliza MVC (Modelo Vista Controlador), tiene su propia forma de trabajo en este punto, con variantes del MVC clásico como la capa de abstracción de base de datos, el controlador frontal y las acciones.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Una potente línea de comandos que facilitan generación de código, lo cual contribuye a ahorrar tiempo de trabajo.

Por último, con miras a un desarrollo sostenible que Symfony se distribuye bajo licencia Open Source MIT, que no impone restricciones y permite el desarrollo de código abierto, así como aplicaciones propietarias.

Servidor web. Apache v2.4

El servidor HTTP Apache es un servidor web HTTP de código abierto, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual según la normativa RFC 2616. Apache consistía solamente en un conjunto de parches a aplicar al servidor de NCSA. (Netcraft 2009)

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido. Apache es el servidor HTTP más usado, este jugó un papel fundamental en el desarrollo de la World Wide Web, siendo el servidor empleado en el 70% de los sitios web en el mundo. En 2009, se convirtió en el primer servidor web que alojó más de 100 millones de sitios web. (Netcraft 2009)

Por las características antes mencionadas y por la necesidad de tener un servidor que brinde rapidez en la navegación y seguridad de los datos con los que se trabaja es seleccionado Apache 2.4 como el servidor web.

1.6. Patrones arquitectónicos

Los patrones arquitectónicos, o patrones de arquitectura, también llamados arquetipos ofrecen soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor. (Pressman 2010)

Ejemplos de patrones arquitectónicos:

- Programación por capas
- Tres niveles
- Arquitectura dirigida por eventos, Presentación-abstracción-control
- Peer-to-peer
- Arquitectura orientada a servicios
- Modelo Vista Controlador

Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación 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. Este patrón de arquitectura de software 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 (González 2012).

A continuación, se muestra el flujo de la información en el patrón MVC.

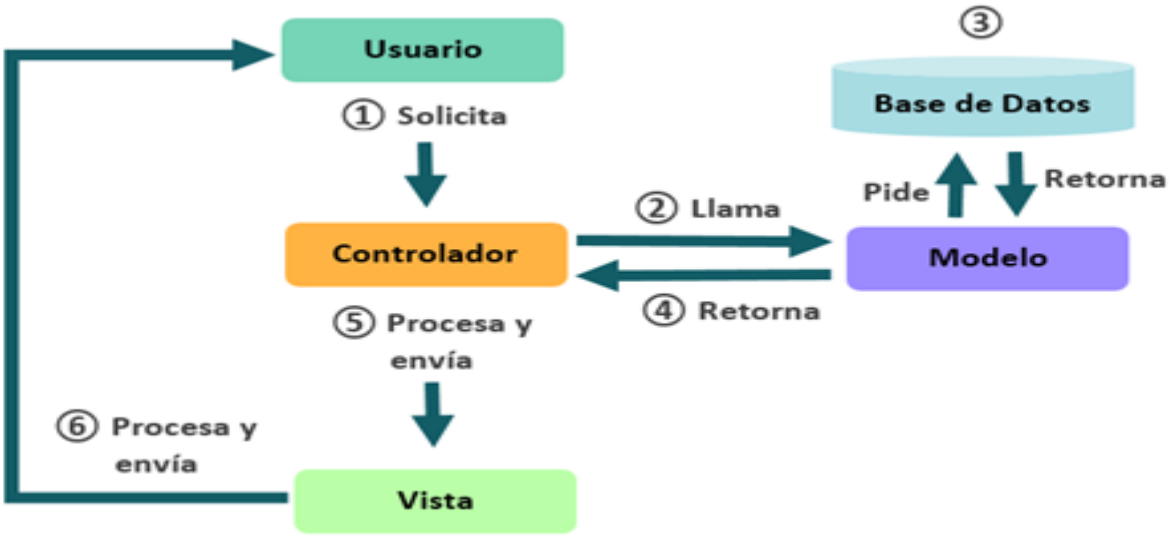

Figura 5 Modelo Vista Controlador

Fuente: (González 2012)

1.7. Patrones de diseño

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño. Entre los patrones de diseño más reconocidos y aplicados en el desarrollo de software, se encuentran los Patrones Generales de Software para Asignar Responsabilidades (GRASP por sus siglas en inglés) y Patrones Banda de los Cuatro (GoF por sus siglas en inglés) (Hamon 2014)

1.7.1. Patrones GRASP

En diseño orientado a objetos, GRASP son patrones generales de software para asignación de responsabilidades, aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software. (Larman 2003)

Entre los patrones GRASP tenemos el **experto en información** el cual es el principio básico de asignación de responsabilidades. Indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento). (Larman 2003)

El patrón **creador** ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases.

La nueva instancia deberá ser creada por la clase que:

- Tiene la información necesaria para realizar la creación del objeto, o
- Usa directamente las instancias creadas del objeto, o
- Almacena o maneja varias instancias de la clase
- Contiene o agrega la clase.

Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización. (Larman 2003)

El patrón **controlador** es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. (Larman 2003)

El patrón **alta cohesión** nos dice que la información que almacena una clase debe ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. (Larman 2003)

El patrón **bajo acoplamiento** es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. (Larman 2003)

1.7.2. Patrones GoF

Estos patrones surgen a raíz del trabajo de un grupo de autores conocido como el *Gang of Four*, formado por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. El grupo GoF agrupó los patrones en tres grandes categorías de acuerdo a su propósito. Los patrones de diseño pueden tener propósito creacional, estructural o de comportamiento (Gamma et al. 1994).

Patrones creacionales: se encargan de las formas de crear instancias de objetos. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados (Gamma et al. 1994).

Patrones estructurales: tratan la composición de clases y objetos, se ocupan de cómo estos se agrupan para formar estructuras más grandes, permitiendo que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos (Gamma et al. 1994).

Patrones de comportamiento: nos ayudan a definir la comunicación e interacción entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.(Gamma et al. 1994)

1.8. Métricas para evaluar el diseño de software

Las métricas son la maduración de una disciplina, que según Pressman, van a ayudar a la evaluación de los modelos de análisis y de diseño, en donde proporcionarán una indicación de la complejidad de diseños procedimentales y de código fuente, y ayudaran en el diseño de pruebas más efectivas; es por eso que propone un proceso de medición, el cual se puede caracterizar por cinco actividades (Pressman 2010):

1. Formulación: La obtención de medidas y métricas del software apropiadas para la representación de software en cuestión.
2. Colección: El mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
3. Análisis: El cálculo de las métricas y la aplicación de herramientas matemáticas.
4. Interpretación: La evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
5. Realimentación: Recomendaciones obtenidas de la interpretación de métricas técnicas trasmitidas al equipo de software.

A continuación, se presentan distintos grupos de métricas que por ser los más conocidos, representativos o utilizados, son los que se han considerado para este trabajo.

- **Métricas CK Chidamber y Kemerer:** Son métricas orientadas a clases: clases individuales, herencia y colaboraciones. Es uno de los conjuntos de métricas más referenciado. (Chidamber, Kemerer 1994)
- **Métricas de Halstead:** son un conjunto de medidas primitivas que determinan el tamaño del software asumiendo que el programa está compuesto por un conjunto de elementos que se clasifican en operadores u operandos. (Patton, 2006)

- **Líneas de código:** es el atributo más usado para interpretar el tamaño de un programa. No existe una definición exacta de qué se considera realmente una línea de código pudiendo excluir o incluir las líneas de comentario, las líneas declarativas y las líneas en blanco. (Chidamber, Kemerer 1994)
- **Métricas de Li Henry:** consideran cinco de las métricas de Chidamber y Kemerer: WMC, DIT, NOC, RFC, LCOM. (Arregui, 2005)
- **Métricas de Lorenz y Kidd (1994):** dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. (González 2001)

1.9. Pruebas de software

Las pruebas de software (en inglés software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada. Es una actividad más en el proceso de control de calidad. Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo. (Barrientos 2014)

1.9.1. Métodos de prueba

Prueba de Caja blanca: se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El ingeniero de pruebas escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados. Su cometido es comprobar los flujos de ejecución dentro de cada unidad (función, clase, módulo, etc.) pero también pueden probar los flujos entre unidades durante la integración, e incluso entre subsistemas, durante las pruebas de sistema.
Las principales técnicas de diseño de pruebas de caja blanca son (Pressman 2010):

- Prueba del camino básico: el método del camino básico permite obtener una medida de la complejidad de un diseño procedimental, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.
- Prueba de bucles: es un método que se centra en la validez de la construcción de los bucles.
- 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.

- 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.

Pruebas de Caja negra: en teoría de sistemas y física, se denomina Caja Negra a aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. Por tanto, de una caja negra deben estar muy bien definidas sus entradas y salidas, es decir, su interfaz. La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca, como (Pressman 2010), por ejemplo:

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores en estructuras de datos o en acceso a las Bases de Datos externas
- Errores de rendimiento
- Errores de inicialización

Técnicas de pruebas de caja negra:

Partición de equivalencia: es una técnica de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos de los que pueden derivarse casos de prueba. El diseño de casos de prueba para la partición de equivalencia se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia, representa un conjunto de estados válidos o inválidos para condiciones de entrada. Por lo general, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición booleana.

Análisis de valor de frontera: un mayor número de errores ocurre en las fronteras del dominio de entrada y no en el “centro”. Por esta razón es que el análisis de valor de frontera se desarrolló como una técnica de prueba. El análisis de valor de frontera conduce a una selección de casos de prueba que revisan los valores de frontera (Pressman 2010).

1.10. Conclusiones parciales

El estudio de los diferentes sistemas de gestión jurídicos nacionales e internacionales demostró que no realiza la gestión de plantillas de documentos jurídicos afectando la usabilidad del sistema, fomentando la necesidad para el proyecto SIGEF de implementar dicho proceso.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se introduce la información necesaria para la comprensión de los pasos seguidos por el grupo de desarrollo del proyecto SIGEF, en cuanto a la selección de las tecnologías y herramientas a utilizar. Esta selección garantiza un desarrollo óptimo del proyecto, evita ambigüedades y proporciona la selección de herramientas funcionales, actualizadas, eficientes y multiplataforma que se integren a las necesidades actuales de la migración a software libre y de esta forma disminuir gastos de recursos. Además, son herramientas que poseen una gran cantidad de información para aprendizaje y comunidades de desarrollo en Internet que permiten el ahorro de tiempo por concepto de adiestramiento.

Se define la infraestructura necesaria para transitar por las fases de modelado del negocio, implementación y prueba con el fin de dar solución a la problemática y cumplir el objetivo trazado, para esto se realizó un estudio que permitió fundamentar el uso de:

- La metodología AUP-UCI, que se considera oportuna para el desarrollo de la solución atendiendo a su estandarización de procesos para el cumplimiento de CMMI-Dev nivel 2 y su institucionalización en la actividad productiva de la UCI.
- Los patrones arquitectónicos y de diseño que permitirán obtener una solución más eficiente.
- Las distintas pruebas que mejoran la calidad de la solución.

CAPÍTULO 2. DESCRIPCIÓN DE LA PROPUESTA

2.1. Introducción.

En el presente capitulo se contribuye al cumplimiento del objetivo general de la investigación. Se realiza una breve descripción de la solución propuesta, se identifican las necesidades del cliente. Se sigue la estructura de la metodología AUP-UCI para generar los artefactos correspondientes a cada disciplina como son los diagramas de procesos de negocio, la descripción de procesos de negocio, las descripciones de requisitos por procesos, así como interfaces para lograr una representación visual de la solución, el modelo conceptual, el diagrama de clases del diseño, el diagrama de despliegue y el modelo de datos. Se hace referencia a la especificación de los requisitos funcionales que abarcan los procesos a desarrollar, teniendo en cuenta las restricciones del sistema y se realiza la especificación de los requisitos no funcionales. Además, se pone de manifiesto la aplicación de los patrones tanto arquitectónicos como de diseño.

2.2. Enfoque propuesto

La gestión de plantillas de los documentos de los procesos del SIGEF, es la funcionalidad propuesta, la cual permite a los fiscales editar las plantillas de documentos y acomodarlas al proceso que se estará tratando. En el SIGEF se generan documentos de cada uno de los procesos, los cuales después de realizados son sometidos a varias revisiones, para mejorarlos hasta casi perfeccionarlos, las plantillas de documentos deben ser editables y ajustables a cada uno de los procesos. Para ello como solución se realiza un módulo para modificar las plantillas de documentos.

2.3. Modelado del negocio

El modelado del negocio es una técnica para comprender los procesos de negocio de la organización. Con este flujo de trabajo pretendemos llegar a un mejor entendimiento de la organización donde se va a implantar el producto. (CIDECAME 2018)

Los objetivos del modelado de negocio son:

- Entender la estructura y la dinámica de la organización para la cual el sistema va a ser desarrollado (organización objetivo).
- Entender el problema actual en la organización objetivo e identificar sus potenciales y mejoras.
- Asegurar que los clientes, usuarios finales y desarrolladores tengan un entendimiento común de la organización objetivo.

- Derivar los requisitos del sistema necesarios para apoyar a la organización objetivo.

Para lograr estos objetivos, el modelo de negocio describe cómo desarrollar una visión de la nueva organización, basado en esta visión se definen procesos, roles y responsabilidades de la organización por medio de un modelo de Casos de Uso del negocio y un Modelo de Objetos del Negocio. Complementario a estos modelos se desarrollan otras especificaciones, tales como un Glosario.

A partir de la metodología seleccionada se utilizan para el modelado del negocio los artefactos:

1. Descripción de Procesos de Negocio.
2. Modelo Conceptual.

2.3.1. Descripción de Procesos de Negocio

En la tabla 1 se muestra la descripción del proceso gestionar plantilla donde se describe cada elemento para un mayor entendimiento por parte de los desarrolladores.

Tabla 1 Descripción del proceso Gestionar Plantilla

Objetivo	Editar las plantillas de documentos generadas.
Evento(s) que lo genera(n)	Permitir adaptar las plantillas de documentos a las características y especificaciones de cada proceso.
Pre condiciones	1. El usuario debe tener el rol con los permisos necesarios para realizar la gestión de plantillas. 2. Debe existir una plantilla para editar. 3. Existencia de errores en la plantilla de documento analizada.
Responsable	Fiscales que atienden los procesos.
Clientes internos	Fiscal responsable del proceso y fiscal superior.
Clientes externos	N/A
Entradas	Plantillas de documentos.
Flujo de eventos	
Flujo básico	
1. Cargar plantillas	El fiscal selecciona la plantilla que necesita ser editada.
2. Identificar los elementos a editar	El fiscal identifica los elementos a editar.
3. Modificar elementos	El fiscal modifica los elementos para ajustarlos al proceso que está tratando.

4.Guardar plantilla	Se guarda la plantilla después de haberle hecho las revisiones.
5.Finalizar	Se finaliza la edición.
Post-condiciones	
1. Se modificó una plantilla adaptándola al a un proceso específico.	
2. Se guardan los documentos.	
Salidas	
1. Plantilla modificada.	

Fuente: elaboración propia

Diagrama de proceso de negocio

El modelado de procesos de negocio es una representación gráfica del proceso y todos sus pasos. Es parte de una metodología denominada Gestión de Procesos de Negocio y es esencial para el crecimiento de una empresa. Un aspecto clave de la gestión de procesos de negocio es que representa visualmente una secuencia detallada de los flujos de información y las actividades necesarias para realizar un proceso. A continuación, se muestra el diagrama del proceso Gestionar Plantillas.

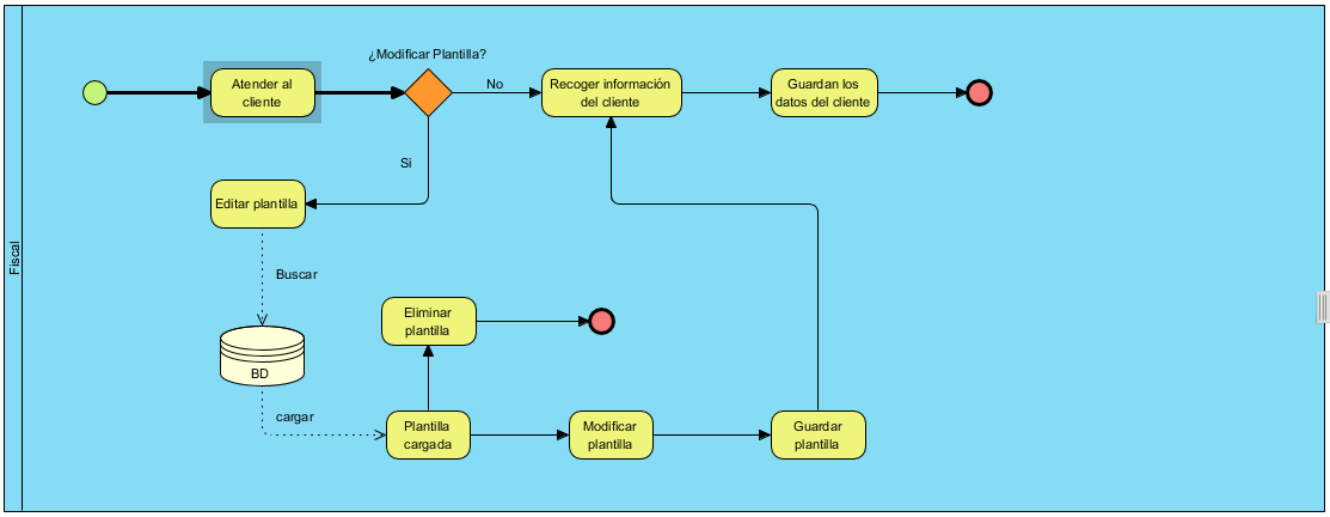

Figura 18 Diagrama del proceso Gestionar Plantilla

Fuente: elaboración propia

2.3.2. Modelo Conceptual

Antes de definir el modelo estático o de clases en UML, es necesario definir el modelo conceptual, el cual muestra los conceptos presentes en el dominio del problema. Un concepto, en este caso, es la representación de cosas del mundo real, no de componentes de software. En él no se definen

operaciones (o métodos). Es fundamental considerar que el modelo conceptual es un proceso abstracto para desarrollar una vista alternativa de cómo funciona el sistema. Se construye en términos de lo que debe entrar al sistema. El ejemplo siguiente pertenece al módulo Editar.

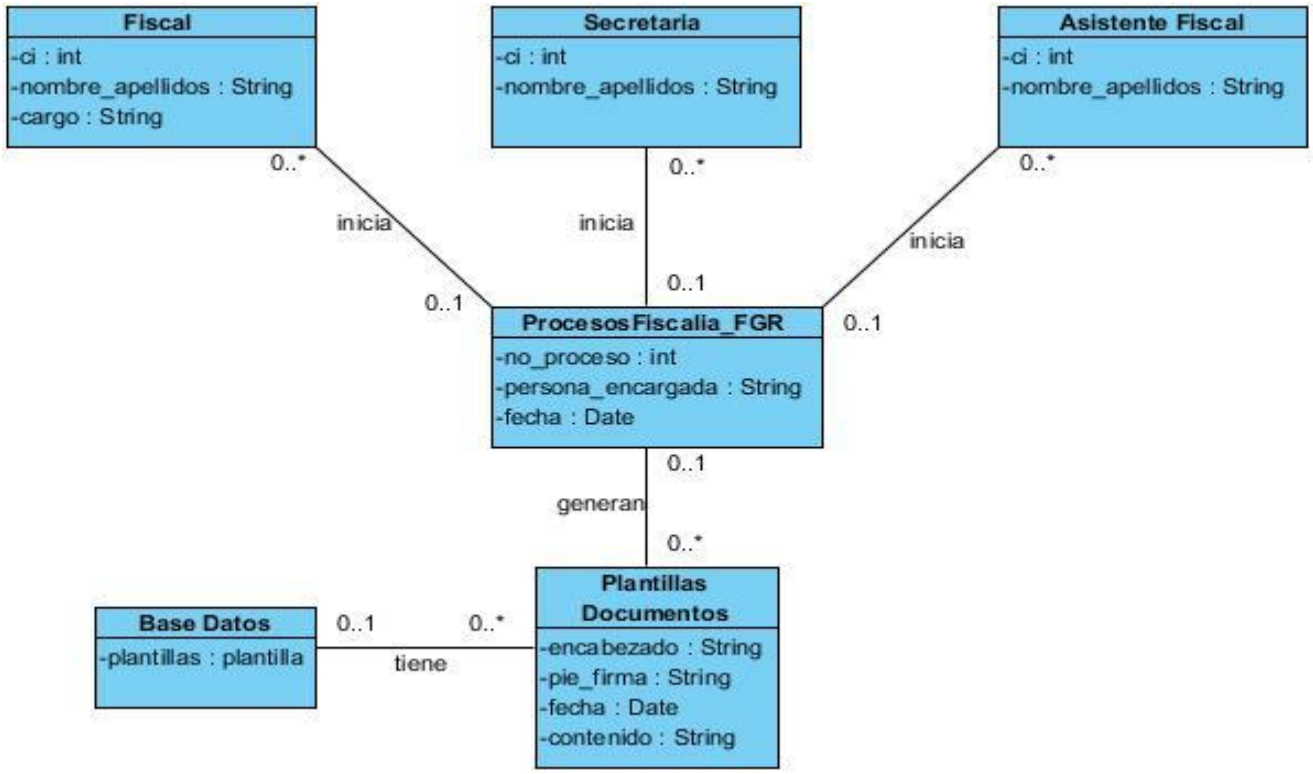

Figura 19 Modelo conceptual

Fuente: elaboración propia

2.4. Requisitos

La ingeniería de requisitos de software consiste en el proceso que permite identificar los servicios y restricciones que formarán un sistema de software. Es la rama de la *ingeniería del software* o *ingeniería de sistemas* que se encarga de la realización de actividades en el intento de entender las necesidades exactas de los usuarios de un sistema y traducir éstas a precisas funciones y acciones que subsecuentemente serán usadas en el desarrollo del sistema. (Pressman 2010).

Mediante el empleo de técnicas de obtención de requisitos como la entrevista con el cliente, la tormenta de ideas y la observación, se definen previamente al trabajo los requisitos que debe cumplir el sistema, dejando documentado los requisitos no funcionales, funcionales y sus especificaciones.

2.4.1. Requisitos no funcionales

A continuación, en la tabla 2, se exponen los requisitos no funcionales identificados con los que debe cumplir la propuesta de solución.

Tabla 2. Requisitos no funcionales

Requisitos no funcionales	
RnF	**Usabilidad**
1	Diseño sencillo, con pocas entradas, permitiendo que no sea necesario mucho entrenamiento para que los usuarios puedan utilizar el sistema.
2	El sistema ofrece una interfaz fácil de operar para el cliente.
3	El sistema tiene que mantener la línea de diseño establecida para la institución que mantiene la uniformidad y representatividad de la misma.
Funcionabilidad	
4	Todo uso de las funcionalidades del sistema requiere la autenticación de los usuarios.
5	El sistema concederá acceso a cada usuario autenticado, solo a las funciones que le estén permitidas, de acuerdo a la configuración del sistema.
Disponibilidad	
6	El sistema estará disponible en todo momento.
7	Disponibilidad de los casos asignados desde cualquier parte del país.
Seguridad	
8	La seguridad se establecerá por roles que se les asignarán a los usuarios que interactúen con el sistema.
9	El sistema mantendrá en todo momento las trazas que se corresponden con las diferentes situaciones críticas que puedan ocurrir.
Requerimientos de Software y Hardware	
10	Máquinas clientes: 1 Gb de memoria RAM o superior. 40 Gb de disco duro o superior.

	Pentium a 1.2 GHz de velocidad de procesamiento o superior. Tarjeta de red. Navegador Web, Chrome o Firefox v50 o superior.
11	Máquinas para servidor aplicación: Sistema operativo Linux PgAdmin III Apache 2.4 Mínimo 4 Gb de RAM o superior. Core i3 4ta generación o superior. Almacenamiento 1 Tb.

Fuente: elaboración propia

2.4.2. Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. En la tabla 3 se registran los requisitos capturados y una descripción de cada uno de ellos.

Tabla 3. Requisitos funcionales

No.	Nombre	Descripción
RF1	Autenticarse en el sistema.	Ingresar el usuario y la contraseña del fiscal para acceder al sistema.
RF2	Crear plantillas de documentos.	Crear la plantilla de documento que se desea utilizar para un documento en específico.
0RF3	Guardar plantillas de documentos.	Guardar las plantillas de documentos de cada uno de los procesos en la BD.
RF4	Listar plantillas de documentos.	Devolver todas las plantillas de documentos creadas.
RF5	Editar las plantillas de documentos.	Permitir editar las plantillas de documentos.

RF6	Modificar estructura de la plantilla de documentos.	Permitir modificar los componentes que forman la plantilla de documento.
RF7	Eliminar las plantillas de documentos.	Permitir eliminar las plantillas de documentos.
RF8	Crear atención al cliente.	Guardar los datos de una atención al cliente.
RF9	Listar atenciones a los clientes.	Mostrar en un listado todas las atenciones creadas.
RF10	Crear un documento.	Crear un documento a partir de los datos de la atención al cliente.
RF11	Mostrar la atención al cliente.	Mostrar la atención al cliente con el documento asociada a ella.
RF12	Visualizar el documento.	Visualizar el documento asociado a una atención.
RF13	Editar documentos.	Permitir editar los documentos generados a partir de una atención.
RF14	Modificar texto del documento.	Permitir modificar el texto de los documentos.
RF15	Modificar encabezado del documento.	Permitir modificar el encabezado de los documentos.
RF16	Modificar pie de firma del documento.	Permitir modificar el pie de firma de los documentos.
RF17	Guardar nuevo documento.	Guardar el nuevo documento editado anteriormente en la base de datos.

Fuente: elaboración propia

2.4.3. Validación de requisitos:

Se validan los requisitos, con el objetivo de comprobar que el sistema cumple con los requisitos funcionales y no funcionales especificados.

Técnicas de validación de requisitos:

- Revisiones formales de requisitos: se realizaron revisiones formales de cada requisito, por parte del cliente y del equipo de desarrollo, validando que la interpretación de cada una de las descripciones no presente omisiones, errores ni ambigüedades**.**
- Construcción de prototipos: se realizó un modelo ejecutable utilizando el componente para que los clientes puedan experimentar con él y determinar si cumple sus necesidades**.**

Se aplicó la métrica Calidad de la Especificación (CE), para medir la calidad de los requisitos de software. Para calcular el total de requisitos de un software se utiliza la siguiente fórmula:

Nr = Nf + Nnf

Nr = 17 + 11

Nr = 28

Nr: total de requisitos de software.

Nf: cantidad de requisitos funcionales.

Nnf: cantidad de requisitos no funcionales.

Para el cálculo de la Especificidad de los Requisitos (ER) o ausencia de ambigüedad, se realiza la operación Q1 = Nui / Nr. Los valores de las variables fueron sustituidos en la ecuación teniéndose en cuenta que, de los requisitos especificados para el desarrollo del componente, los RF5 y RF6 causaron contradicción en sus interpretaciones. Por tanto, la variable Q1 toma el siguiente valor:

Q1 = Nui / Nr

Q1 = 26 / 28

Q1 = 0.93

Nui: número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas.

Mientras más cerca esté el valor Q1 de 1, menor es la ambigüedad. El resultado anterior muestra que el 93 % de los requisitos es entendible.

2.4.4. Descripción de Requisitos por Proceso

Existen tres formas de encapsular los requisitos [Casos de Uso del Sistema (CUS), Historias de Usuario (HU) y Descripción de Requisitos por Proceso (DRP)], agrupados en cuatro escenarios condicionados por el Modelado de negocio (Sánchez 2015).

Siguiendo la metodología seleccionada para la investigación y el escenario escogido se utiliza la Descripción de Requisitos por Proceso para encapsular los requisitos. La tabla 4 muestra la descripción de requisito por proceso del requisito funcional Modificar Plantillas

Tabla 3 Descripción del requisito Editar plantillas de documento en el módulo Editar.

Precondiciones		El proceso debe haber finalizado	
Flujo de eventos			
Flujo básico Modificar plantilla			
1.	Seleccionar plantilla.		
2.	Seleccionar acción de editar plantilla.		
3.	Editar componentes de la plantilla.		
4.	Modificar plantilla.		
5.	Guardar plantilla.		
Pos-condiciones			
1.	Se modificó una plantilla.		
2.	Se generó una plantilla de documento nueva.		
3.	Se obtiene una plantilla que no contiene errores.		
Flujos alternativos			
Flujo alternativo Crear plantilla		No existe la plantilla	
1	Crear plantilla.		
Pos-condiciones			
1	Se creó una plantilla de documento nueva.		
Validaciones			
1	Tipo de proceso, área		
Conceptos	**Plantilla**	Visibles en la interfaz: Botón editar. Botón guardar. Botón volver a la lista. Utilizados internamente: N/A	
Requisitos especiales	N/A		
Asuntos pendientes	N/A.		

Fuente: elaboración propia

Toda descripción de requisitos por proceso se encuentra acompañada por un prototipo de interfaz gráfica. A continuación, se muestra el prototipo del proceso Gestionar plantilla en el módulo Editar.

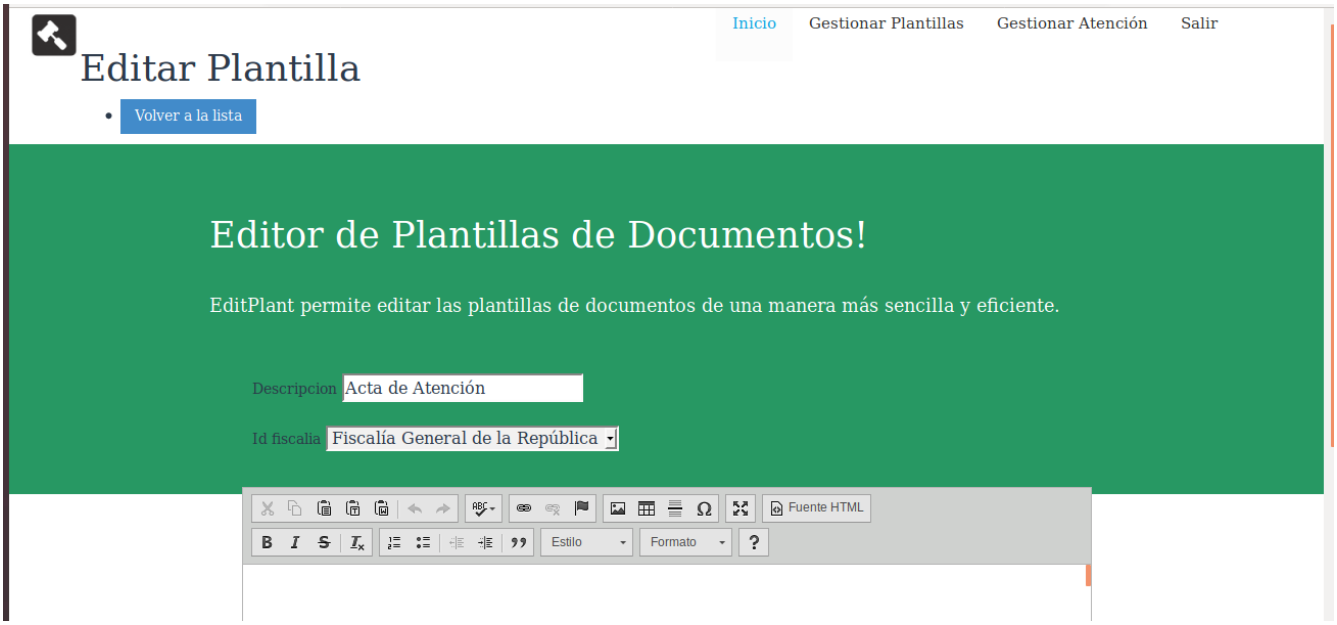

Imagen 1 Prototipo de Gestor de plantillas.

Fuente: elaboración propia.

2.5. Análisis y diseño

El diseño de software es el proceso para la planificación de una solución de software. Este proceso es, por regla general, necesario para que los programadores puedan manejar la complejidad que la mayoría de los programas informáticos poseen y para disminuir el riesgo de desarrollos erróneos. El diseño propuesto tiene que cumplir a cabalidad con los requerimientos del sistema, ya que es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado. Debe ser capaz de facilitar las mejoras del software, tiene que ser entendible por otros profesionales de la especialidad, de manera que permita la comprobación del sistema fácilmente (Larman 2004).

2.5.1. Diagrama de clases del diseño

El diagrama de clases del diseño representa las clases del sistema con sus relaciones estructurales. En el caso de las aplicaciones web y del patrón arquitectónico MVC, se representan las colaboraciones que existen entre las clases en las distintas capas (Zapata 2007).

Con el fin de lograr una mejor comprensión de la aplicación, se modela en la figura 9 el diagrama de clases del diseño del componente desarrollado.

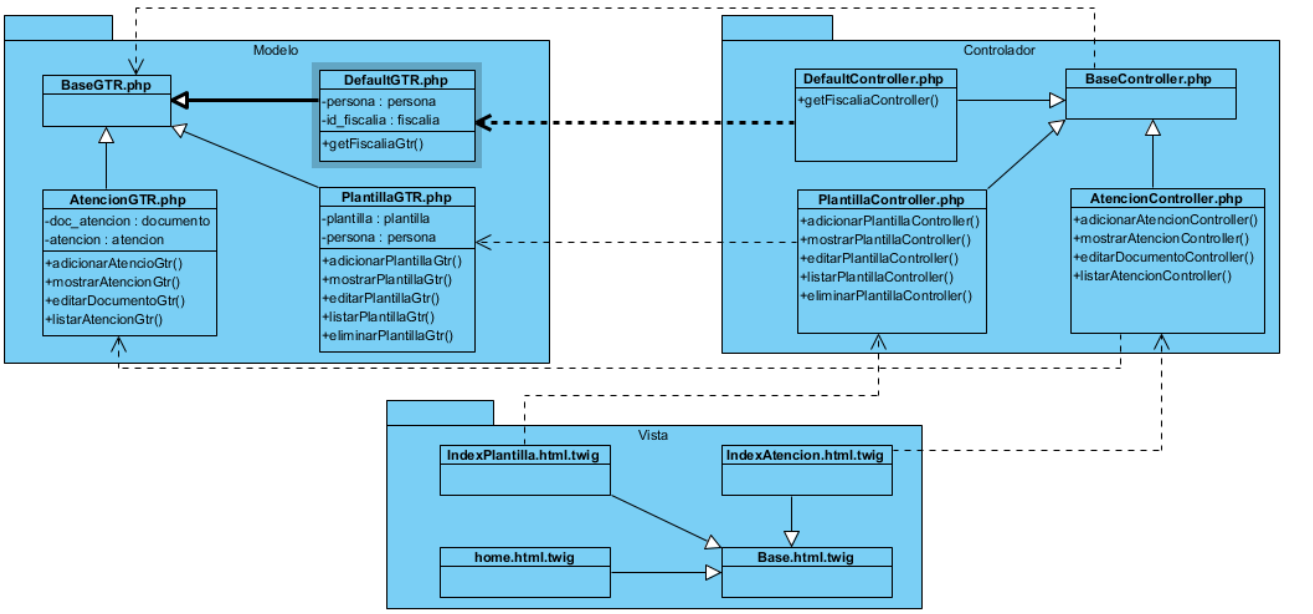

Figura 8 Diagrama de Clases del módulo Gestionar Plantilla.

Fuente: elaboración propia

2.5.2. Modelo de datos

Para representar la estructura de la base de datos del módulo gestionar plantilla se presenta el modelo de datos confeccionado anteriormente por el equipo de trabajo, permitiendo la representación lógica de los datos procesados por el sistema. Este modelo fue construido utilizando la técnica de modelado de datos Entidad-Relación-Atributo, mostrando las entidades de datos, sus atributos asociados y las relaciones entre las entidades. A continuación, se muestra una imagen del modelo de datos del módulo.

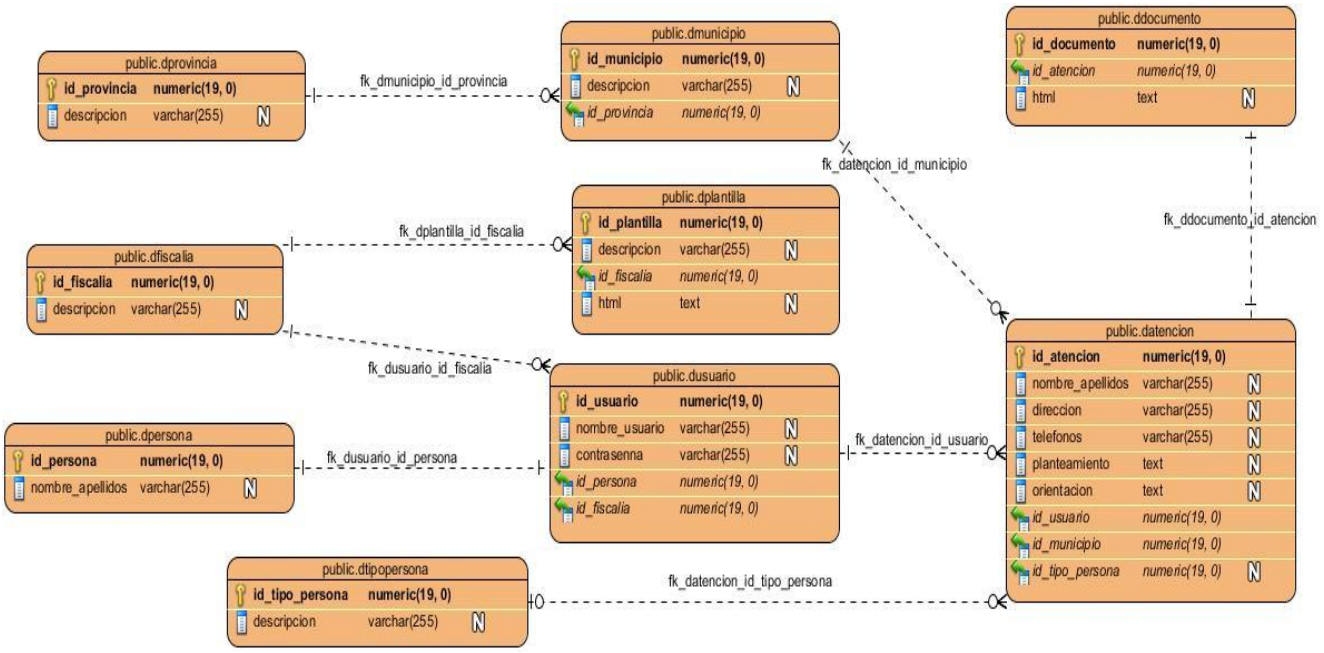

Figura 9 Modelo de datos del módulo Gestionar Plantilla.

Fuente: elaboración propia

2.5.3. Patrón arquitectónico

Para el desarrollo de la aplicación se emplea el patrón arquitectónico Modelo-Vista-Controlador, facilitado por el marco de trabajo seleccionado *(Symfony3*). En la aplicación las tres capas que lo definen son empleadas de la siguiente forma:

- El modelo representa la información o las clases con las que trabaja la aplicación. En la aplicación son las clases gestoras como PlantillaGtr las cuales separan la lógica del negocio del resto de la aplicación, facilitando la reutilización de código.
- 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 de esto es index.html.twig, la cual se encuentra almacenada en el directorio *App/AppBundle/Resources/view/dplantilla.*
- El controlador es la parte del 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 propuesta de solución todas las solicitudes son gestionadas por el controlador PlantillaController.php.

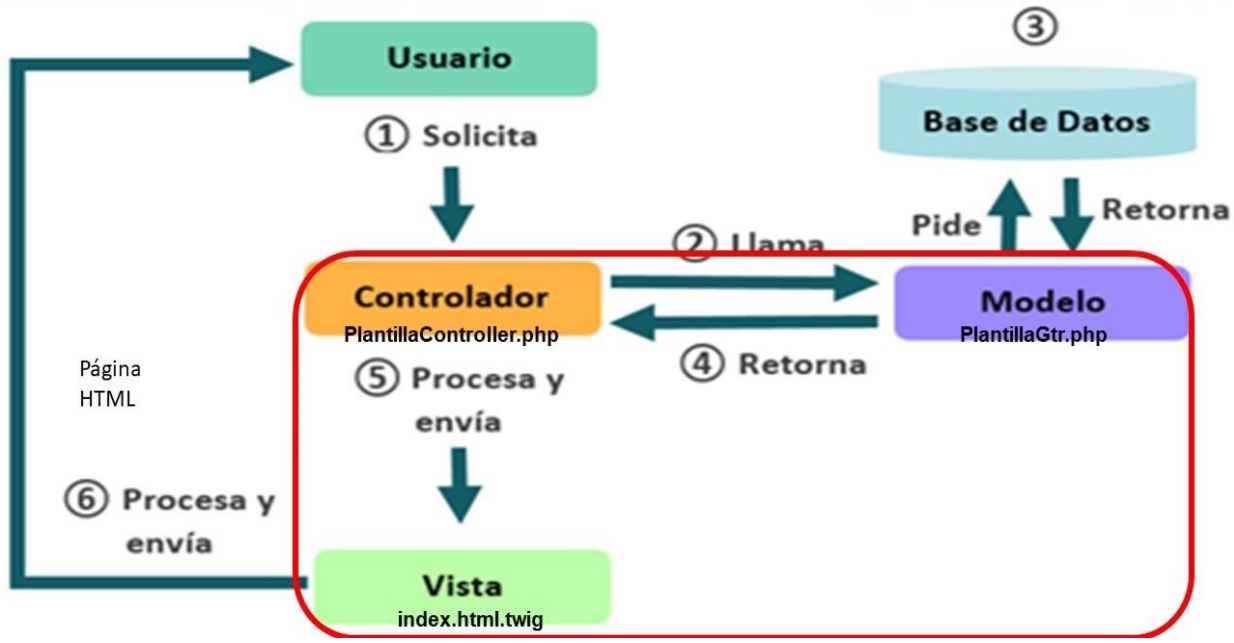

Figura 10 Patrón Arquitectónico Modelo-Vista-Controlador.

Fuente: elaboración propia

2.5.4. Patrones de diseño

Para el desarrollo de la solución se utilizaron un conjunto de patrones de diseño que se explican a continuación:

Patrones GRASP:

- Experto en información: se utiliza para asignar responsabilidades al experto de la información, es decir, a la clase que tiene la información necesaria para llevar la tarea a cabo. Esto provoca un encapsulamiento de la información y, por ende, el bajo acoplamiento, además de un comportamiento distribuido entre las clases, es decir, clases más cohesivas. El patrón Experto se evidencia en la clase *Plantilla.php* para el mapeo de la base de datos.
- Creador: el patrón creador permite que una clase B tenga la responsabilidad para crear una instancia de la clase A. La clase *PlantillaGtr.php* es una clase creadora dado que en esta 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.
- Alta cohesión: se encuentra fundamentalmente evidenciada en las clases controladoras como *PlantillaController.php*, la cual tiene la responsabilidad de definir y realizar todas las acciones

que son solicitadas, conjuntamente con otras clases colabora 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: se evidencia en todas las clases controladoras que heredan de *BaseController.php* para lograr un bajo acoplamiento entre ellas y así crear un sistema flexible a grandes cambios, de esta manera se alcanza una mejor compresión de las clases aisladas, se facilita la reutilización de código y no afectan los cambios en otros componentes. Ejemplo de esto es la clase *AtencionController.php.*
- Controlador: todas las peticiones web son manejadas por un solo controlador frontal (*app.php*), que es el punto de entrada único de 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 de un módulo con la URL entrada por el usuario.

Patrones GoF:

- Decorador: este añade funcionalidades a una clase dinámicamente, se aplica en la generación de las vistas, adicionando elementos reutilizables y comunes a las vistas que heredan de ella, como el menú, el pie de página, el encabezado, entre otros elementos. Un ejemplo de esto es el archivo *base.html.twig* que contiene el código común para decorar todas las demás páginas. La utilización del patrón decorador se ve evidenciada en las clases *home.html.twig* y *newplantilla.html.twig.*
- Instancia única: garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el controlador frontal *sfWebFrontController* hay una llamada a *sfContext::getInstance()* y se encarga de enrutar todas las peticiones que se hagan a la aplicación.
- Mando: este patrón se observa en la clase *sfWebFrontController*, en el método *dispatch().* Esta clase está por defecto y es la encargada de establecer el módulo y la acción que se va a usar según la petición del usuario. Este patrón se aplica además en la clase *sfRouting*, que está desactivada por defecto y procede según las necesidades del administrador del sistema donde se aplique el *framework*, la cual se puede activar o desactivar. En este método es parseada la URL con el objetivo de precisar los parámetros de la misma y de esta forma saber el método *Actions* que debe responder a la petición.
- Registro: este patrón es un medio sencillo y eficiente de compartir datos y objetos en la aplicación sin la necesidad de preocuparse por conservar numerosos parámetros o hacer uso

de variables globales. Este patrón se aplica en la clase *sfConfig*, que es la encargada de acumular todas las variables de uso global en el sistema.

2.6 Estándares de codificación

Los estándares de código, son parte de las llamadas buenas prácticas o mejores prácticas, estas son un conjunto no formal de reglas, las cuales bien aplicadas pueden incrementar la calidad de tu código notablemente, haciendo más fácil para cualquier otro miembro del equipo la comprensión del código, dichas reglas son un paradigma de programación que busca reducir el número de decisiones que el desarrollador tiene que tomar al momento de escribir su código (Merkury 2017). El estándar utilizado en la codificación fue el siguiente:

Notación **CamelCase:** es un estilo de escritura que se aplica a frases o palabras compuestas. Donde estas palabras no llevan espacios entre sí. Existen dos tipos de *CamelCase*, a continuación, se mencionan (Loera 2014):

- ***UpperCamelCase:*** cuando la primera letra de cada una de las palabras es mayúscula, esta notación es utilizada para nombrar las clases. Ej. PlantillaController.php
- ***LowerCamelCase:*** igual que UpperCamelCase con la excepción de que la primera letra de la primera palabra es minúscula, esta notación es utilizada para nombrar los métodos. Ej. newPlantillaAction.

2.7 Conclusiones parciales

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

- El empleo de técnicas de obtención de requisitos como la entrevista con el cliente, la observación y la tormenta de ideas permitieron capturar los requisitos funcionales y no funcionales para la implementación del componente de gestión de plantillas en el Sistema de Gestión Fiscal.
- La modelación del proceso de negocio permitió tener una mejor comprensión de la gestión de plantillas de documentos del proceso de atención al ciudadano.
- La etapa de diseño brindó una visión más profunda de la aplicación.
- La definición del patrón arquitectónico MVC y los patrones de diseño utilizados, permitieron establecer las bases para fomentar la reutilización de código y las buenas prácticas de programación entre los desarrolladores durante la fase de implementación, así como disminuir el impacto de los cambios futuros en el código fuente.

CAPÍTULO 3. VALIDACIÓN DE LA PROPUESTA

3.1. Introducción

Los softwares deben pasar por un conjunto de pruebas las cuales permiten validar el correcto funcionamiento de los mismos. En el presente capítulo se explican los resultados arrojados por las pruebas realizadas a las diferentes funcionalidades. Se emplean métricas para la validación del diseño, se realizan pruebas de aceptación con el cliente para garantizar que el software cumpla con los objetivos de la presente investigación.

3.2. Validación del diseño

Métrica Tamaño Operacional de Clases (TOC)

Para la aplicación de esta métrica se tomarán como muestra 15 clases, de un total de 21 clases, distribuidas en la capa controladora, la vista y el modelo, registrándose un total de 36 métodos con un promedio de 2.4 procedimientos por clase. Esta evalúa los siguientes atributos de calidad: responsabilidad, complejidad de implementación y reutilización.

Tabla 4 evaluación de clases para TOC

No	Subsistema	Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad de implementación	Reutilización
1	Administración	PlantillaController.php	5	Alta	Alta	Baja
2	Plantilla	listarPlantilla.html.twig	1	Baja	Baja	Alta
3	Plantilla	newPlantilla.html.twig	1	Baja	Baja	Alta
4	Plantilla	showPlantilla.html.twig	3	Media	Media	Media
5	Plantilla	editPlantilla.html.twig	5	Alta	Alta	Baja
6	Plantilla	eliminarPlantilla.html.twig	1	Baja	Baja	Alta
7	Administración	atencionController.php	5	Alta	Alta	Baja
8	Atención	listarAtencion.html.twig	1	Baja	Baja	Alta
9	Atención	newAtencion.html.twig	1	Baja	Baja	Alta
10	Atención	showAtencion.html.twig	1	Baja	Baja	Alta
11	Atención	editAtencion.html.twig	1	Baja	Baja	Alta
12	Administración	DefaultController.php	4	Media	Media	Media
13	Administración	Base.html.twig	3	Media	Media	Media

14	Plantilla	PlantillaGtr.php	4	Media	Media	Media
15	Atención	AtencionGtr.php	4	Media	Media	Media

Fuente: elaboración propia

Tabla 5 Umbrales definidos para la métrica TOC

Categoría	Criterio
Responsabilidad	
Baja	< =Prom.
Media	Entre Prom. y 2* Prom.
Alta	> 2* Prom.
Complejidad implementación	
Baja	< =Prom.
Media	Entre Prom. y 2* Prom.
Alta	> 2* Prom.
Reutilización	
Baja	> 2*Prom.
Media	Entre Prom. y 2* Prom.
Alta	<= Prom.

Fuente: elaboración propia

A continuación, se muestran las gráficas con los resultados de las mediciones de los atributos de calidad: responsabilidad, reutilización y complejidad de implementación.

Figura 20 Resultados obtenidos para el atributo Responsabilidad

Fuente: elaboración propia

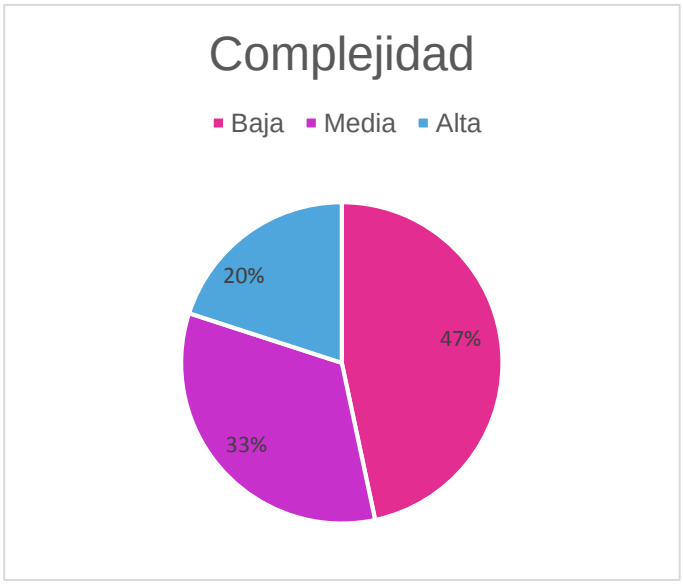

Figura 21 Resultados obtenidos para el atributo Complejidad de Implementación

Fuente: elaboración propia

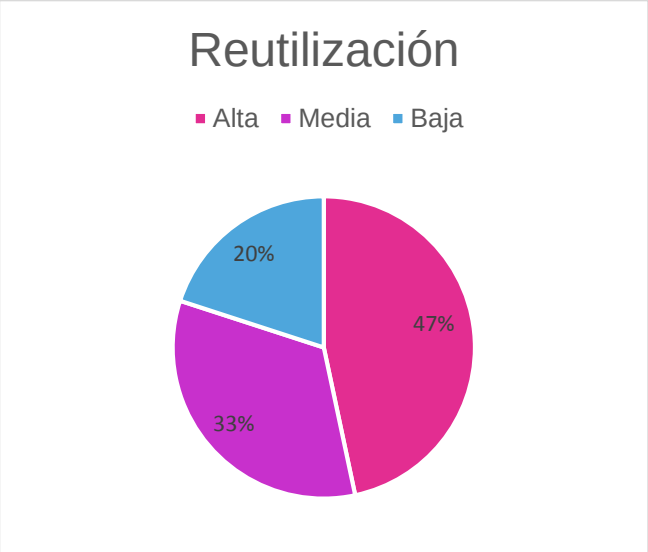

Figura 22 Resultados obtenidos para el atributo Reutilización

Fuente: elaboración propia

La aplicación de la métrica TOC, permite concluir que las clases del diseño de los procesos en el módulo propuesto para el SIGEF, no asumen grandes responsabilidades, posee una baja complejidad de implementación y cuentan con una reutilización alta de código. Por tanto, se puede afirmar que los resultados obtenidos de esta métrica son positivos.

Métrica Relaciones entre Clases (RC)

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

Tabla 6 Evaluación de clases para RC

No	Subsistema	Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mantenimiento.	Reutilización	Cantidad de Pruebas
1	Administración	AplicacionController.php	5	Alto	Alto	Baja	Alta
2	Plantilla	listarPlantilla.html.twig	1	Bajo	Bajo	Alta	Baja
3	Plantilla	newPlantilla.html.twig	1	Bajo	Bajo	Alta	Baja

4	Plantilla	showPlantilla.html.twig	1	Bajo	Bajo	Alta	Baja
5	Plantilla	editPlantilla.html.twig	5	Alto	Alto	Media	Media
6	Plantilla	eliminarPlantilla.html.twig	1	Bajo	Bajo	Alta	Baja
7	Administración	atencionController.php	5	Alto	Alto	Media	Media
8	Atención	listarAtencion.html.twig	1	Bajo	Bajo	Alta	Baja
9	Atención	newAtencion.html.twig	2	Medio	Medio	Alta	Baja
10	Atención	showAtencion.html.twig	1	Bajo	Bajo	Alta	Baja
11	Atención	editAtencion.html.twig	2	Medio	Medio	Alta	Baja
12	Administración	DefaultController.php	6	Alto	Alto	Baja	Alta
13	Administración	base.html.twig	3	Alto	Medio	Media	Media
14	Plantilla	PlantillaGtr.php	1	Bajo	Bajo	Alta	Baja
15	Atención	AplicacionGtr.php	2	Medio	Medio	Alta	Baja

Fuente: elaboración propia

Tabla 7 Umbral definido para la métrica RC

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

Fuente: elaboración propia

A continuación, se muestran las gráficas con los resultados de las mediciones de los atributos: acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización.

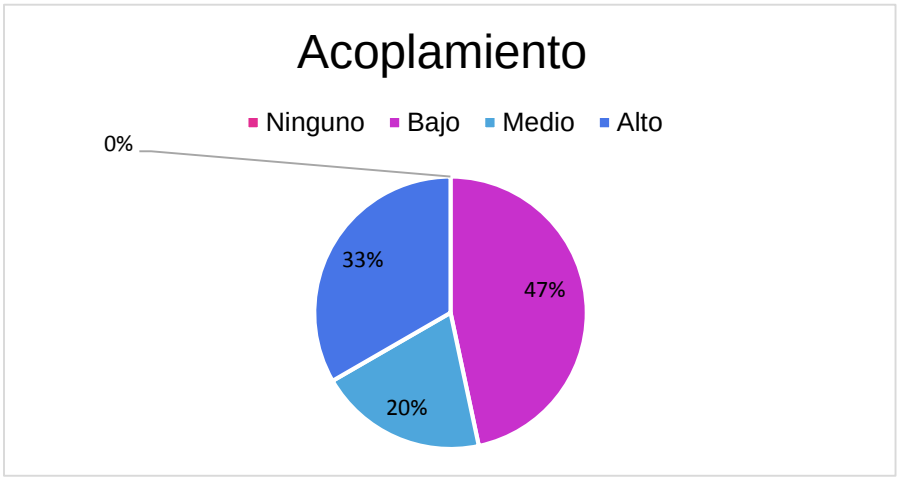

Figura 23 Resultados obtenidos para el atributo Acoplamiento

Fuente: elaboración propia

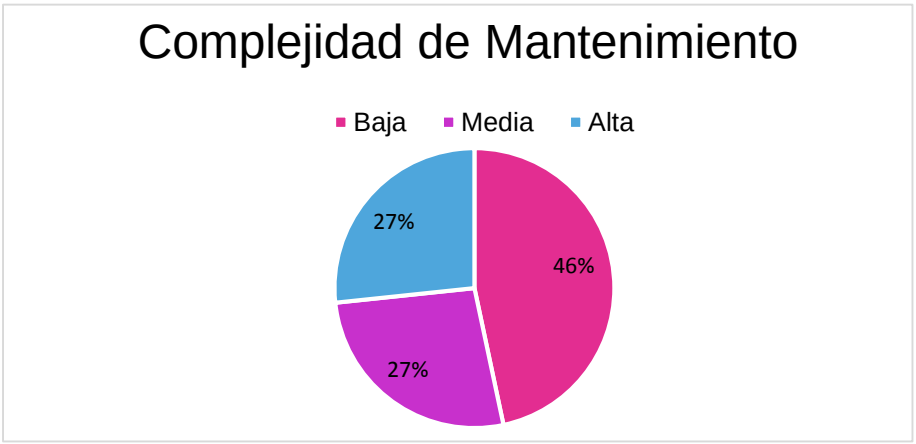

Figura 24 Resultados obtenidos para el atributo Complejidad de Mantenimiento

Fuente: elaboración propia.

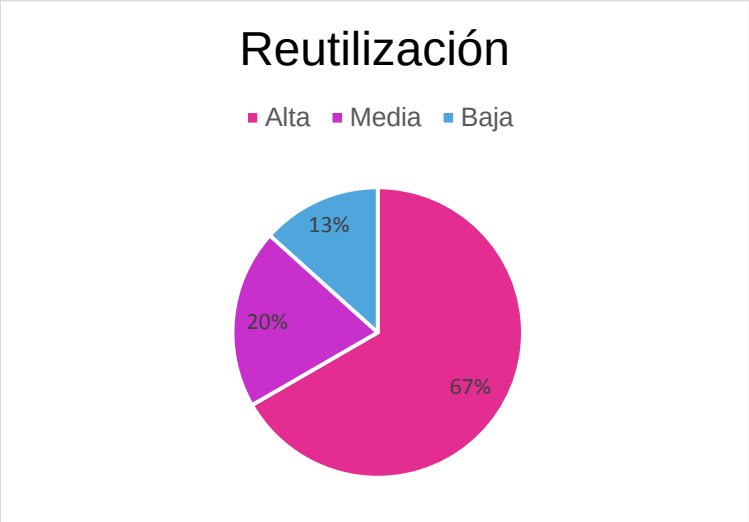

Figura 16 Resultados obtenidos para el atributo Reutilización

Fuente: elaboración propia.

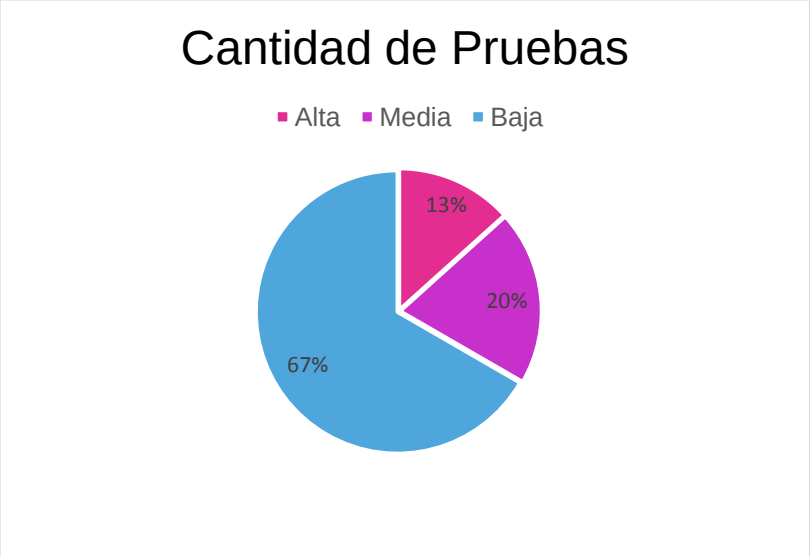

Figura 17 Resultados obtenidos para el atributo Cantidad de Pruebas

Fuente: elaboración propia.

A partir de la aplicación de la métrica RC, se tiene como resultado que los porcientos para los atributos de acoplamiento fueron bajos. Además, los datos obtenidos para los atributos de complejidad de mantenimiento, reutilización y cantidad de pruebas se comportan satisfactoriamente en la mayor parte de las clases. Esto se traduce en un diseño adaptable a cambios, reduciendo esfuerzos en la

implementación y en la aplicación de las pruebas unitarias, lo que conlleva a la conclusión de que la calidad del diseño es buena.

3.3.1. Pruebas de unidad

Las pruebas de unidad enfocan los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software. La relativa complejidad de las pruebas y los errores que descubren, están limitados por el ámbito restringido que se establece para las pruebas de unidad. Las pruebas de unidad por lo general se consideran como adjuntas al paso de codificación. El diseño de las pruebas de unidad puede ocurrir antes de comenzar la codificación o después de generar el código fuente. La revisión de la información del diseño proporciona una guía para establecer casos de prueba. Cada caso de prueba debe acoplarse con un conjunto de resultados esperados (Pressman 2010).

Método de caja blanca

Para la aplicación de este método se utiliza la técnica del Ruta básica, para la cual se hace necesario definir los siguientes pasos para una mayor organización del trabajo:

1. Enumerar las sentencias de código del procedimiento.

```
1  public function newAction(Request $request){
2      $datencion = new Datencion();
       $form = $this->createForm( type: 'AppBundle\Form\DatencionType', $datencion);
       $form->handleRequest($request);
3      if ($form->isSubmitted()) {//&& $form->isValid()
           $em = $this->getDoctrine()->getManager();
           $datencion->setIdUsuario($this->getUser());
           $datencion->setFecha(new \DateTime());
4          $em->persist($datencion);
           $acta_atencion = new Ddocumento();
           $acta_atencion->setIdAtencion($datencion);
           $id_plantilla_acta_atencion = 1;
           $dplantilla = $em->getRepository( className: "AppBundle:Dplantilla")->find($id_plantilla_acta_atencion);
5          if (!is_null($dplantilla)) {
               $twig = clone $this->get('twig');
               $twig->setLoader(new \Twig_Loader_String());
               $dpersona = $em->getRepository( className: "AppBundle:Dpersona")->find($datencion->getIdUsuario()->getIdPersona());
               $html = $twig->render(
6                  $dplantilla->getHtml(),
                   array('ds' => $datencion, 'persona'=>$dpersona));
               $acta_atencion->setHtml($html);
               $em->persist($acta_atencion);
               $em->flush();
7          } else {
               return $this->render( view: 'datencion/new.html.twig', array(
8                  'datencion' => $datencion,
                   'form' => $form->createView(),
               ));
           }
9          return $this->redirectToRoute( route: 'datencion_index', array('idAtencion' => $datencion->getIdatencion()));
10     }
       return $this->render( view: 'datencion/new.html.twig', array(
11         'datencion' => $datencion,
           'form' => $form->createView(),
       ));
12 }
```

Figura 25 Método newAction()

Fuente: elaboración propia

La figura 18 muestra la funcionalidad *newAction()* de la clase *PlantillaController.php,* se escoge este método debido a que es el que por sí solo presenta una cantidad considerable de ciclos y condicionales, a partir de la cual se enumeran las sentencias, contribuyendo así a determinar la cantidad de nodos que tendrá el grafo asociado a este método.

2. A partir del código fuente se dibuja el grafo de flujo asociado.

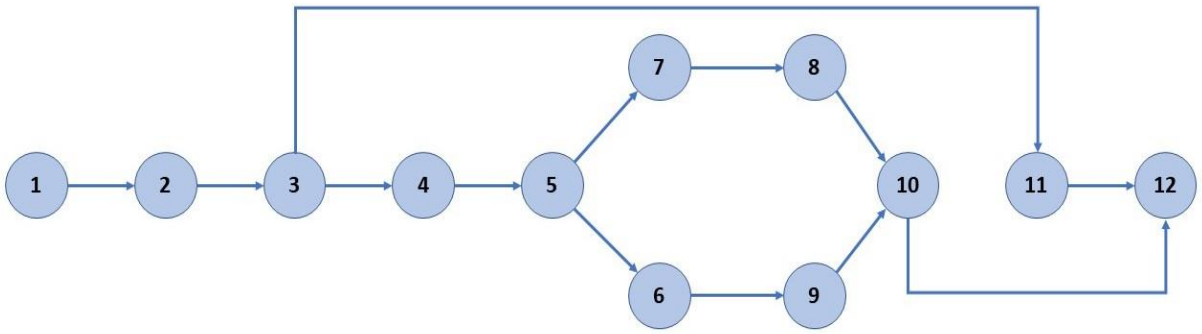

Figura 26 Grafo del método newAction().

Fuente: elaboración propia

El grafo representado permite observar fácilmente la diversidad de caminos que se pueden recorrer al ejecutar el código al cual pertenece.

3. Se calcula la complejidad ciclomática del grafo.

La complejidad ciclomática es una medición de software que proporciona una evaluación cuantitativa de la complejidad lógica de un programa. El valor calculado por la complejidad ciclomática, define el número de rutas independientes del conjunto básico de un programa y le brinda una cota superior para el número de pruebas que debe realizar a fin de asegurar que todos los enunciados se ejecutaron al menos una vez (Pressman 2010).

Para una mayor comprensión del tema se conceptualizarán los términos de nodo, arista, región y nodo predicado. Según (Pressman 2010):

- Nodo: cada círculo del grafo que representa uno o más enunciados de procedimiento.
- Arista: flechas en el grafo que representan flujo de control. Una arista debe terminar en un nodo, incluso si el nodo no representa algún enunciado de procedimiento.

- Región: las áreas acotadas por aristas y nodos se llaman regiones. Cuando se cuentan las regiones, el área afuera del grafo se incluye como región.
- Nodo predicado: cada nodo que contiene una condición y se caracteriza por dos o más aristas que emanan de él.

La complejidad ciclomática se puede calcular de las siguientes formas (Pressman 2010):

1. El número de regiones del grafo de flujo corresponde a la complejidad ciclomática.
2. La complejidad ciclomática V(G) para un grafo de flujo G se define como
   V(G) = E - N + 2
   donde E es el número de aristas del grafo de flujo y N el número de nodos del grafo de flujo
3. La complejidad ciclomática V(G) para un grafo de flujo G también se define como
   V(G) = P + 1
   donde P es el número de nodos predicado contenidos en el grafo de flujo G.

En el grafo representado anteriormente, la complejidad es calculada de las siguientes formas:

1. V(G) = 3
2. V(G) = 13 – 12 + 2 = 3
3. V(G) = 2 + 1 = 3

El cálculo efectuado mediante las tres fórmulas ha arrojado el mismo valor, por tanto, se puede plantear que la complejidad ciclomática del código es 3, lo que significa que existen a lo sumo 3 posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado. Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo.

Tabla 8 Rutas básicas

Número	Ruta básica
1	1-2-3-11-12
2	1-2-3-4-5-6-9-10-12
3	1-2-3-4-5-7-8-10-12

Tabla 9 Casos de prueba del método de caja blanca

Número camino	Descripción	Condición de ejecución	Entrada	Resultados esperados	Resultados	Salida
1	El dato de entrada es un Request con el formulario	La petición se realiza mediante un método POST	$request = isValued (false)	No se crea la plantilla de documento	No se crea la plantilla de documento	**1**
2	El dato de entrada es un Request con el formulario	La petición se realiza mediante un método POST	$request = isValued (true) !isNull($dplantilla)= (true)	Se crea una plantilla de documentos	Se crea la plantilla de documento	**1**
3	El dato de entrada es un Request con el formulario	La petición se realiza mediante un método POST	$request = isValued (true) !isNull($dplantilla)= (false)	No se crea la plantilla de documento	No se crea la plantilla de documento	**1**

La aplicación de las pruebas de caja blanca demostró que el estado real del software coincide con el esperado, comprobándose a través de los casos de prueba desarrollados donde la ejecución de las condiciones y los bucles tuvieron resultados satisfactorios.

Método de caja negra

Con el objetivo de comprobar que las funcionalidades implementadas actúan de acuerdo a las especificaciones de los requisitos, se llevan a cabo pruebas de caja negra a través de la técnica de la Partición de Equivalencia, con el objetivo de reducir la cantidad de diseño de casos de prueba a uno por requisito. Para realizarlas no fue necesario conocer los detalles internos del programa y su objetivo fundamental fue probar que el software se encontraba acorde a los requisitos.

Se ejecutaron un total de 17 Casos de Pruebas (CP), hasta garantizar la eliminación de las no conformidades, logrando que el componente no presente errores. En la tabla 11 se muestra un ejemplo de la descripción de las variables del CP “Editor de Plantillas”, seguida por una gráfica con la información de los resultados obtenidos de las pruebas de Caja Negra por cada iteración (Figura 20)

Tabla 10 Descripción de las variables del CP Editor de Plantillas

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Descripción	Campo de texto	No	Permite cambiar el nombre de la plantilla que se está editando.
2	Fiscalía	Campo de selección	No	Permite seleccionar la fiscalía correspondiente a la plantilla.
3	Plantilla (Html)	Campo de texto	Si	Se carga la plantilla y se permite editarla.

Fuente: elaboración propia

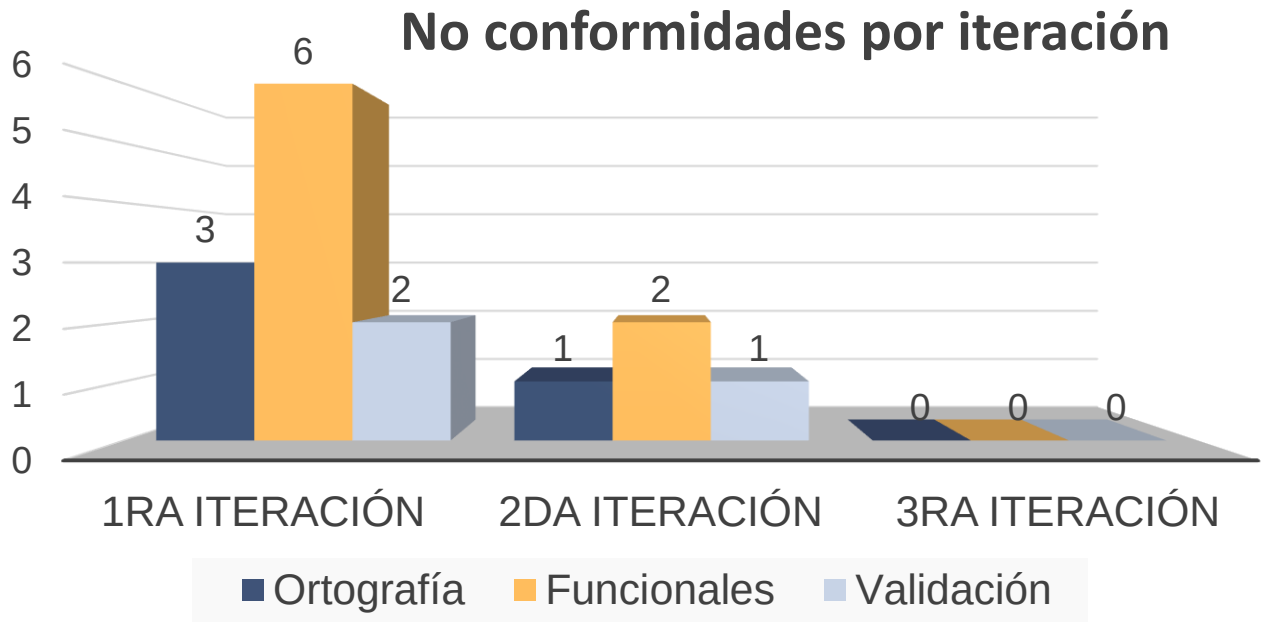

Figura 27 No conformidades por iteración

Fuente: elaboración propia

3.4. Pruebas de liberación

La liberación del software llevada a cabo por el grupo de calidad de CEGEL demostró que el componente cumplía con las necesidades requeridas y estaba listo para ser entregado a la Fiscalía General de la República. Se realizaron un total de 2 iteraciones las cuales arrojaron no conformidades de correspondencia entre el caso de prueba y el sistema, funcionales y recomendación como se muestra en la figura 21. El acta de liberación del componente se puede consultar en el anexo 1.

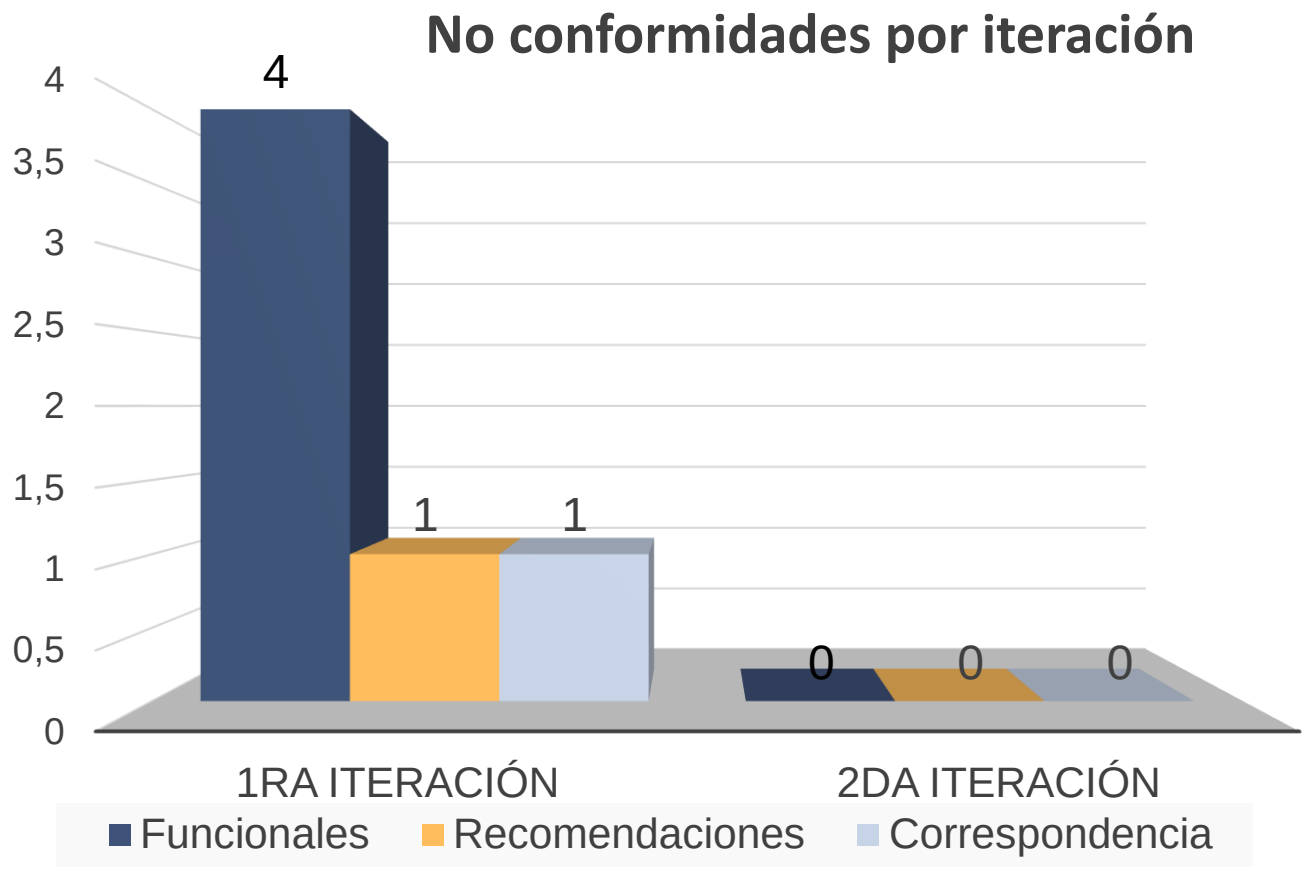

Figura 28 No conformidades por iteración de las pruebas de liberación

Fuente: elaboración propia

3.5. Verificación de las variables de la investigación

Usabilidad

Existen varias definiciones referentes al término usabilidad de las cuales a continuación se mostrarán algunas.

El término “usabilidad”, que deriva del inglés *“Usability”*, es un atributo cualitativo definido comúnmente como la facilidad de uso, ya sea de una página Web, una aplicación informática o cualquier otro sistema que interactúe con un usuario. El concepto generalmente se refiere a una aplicación informática o un aparato, aunque también puede aplicarse a cualquier sistema hecho con algún objetivo particular (Sánchez 2011).

Según la ISO/IEC 9241 “Usabilidad es la eficiencia y satisfacción con la que un producto permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico” (ISO 1998).

Jakob Nielsen, considerado el padre de la usabilidad, la definió como “el atributo de calidad que mide lo fácil de usar que son las interfaces Web”. Es decir, un sitio Web usable es aquél en el que los usuarios pueden interactuar de la forma más fácil, cómoda, segura e inteligente posible (Nielsen 2012).

Atributos de Usabilidad

Una vez abordado el concepto de usabilidad según las distintas fuentes mencionadas, es claro que abordarla implica también revisar una serie de aspectos relacionados con el uso y la manera en que las personas se relacionan con los sistemas que se les ofrecen. Para poder estudiarla se descompone en los siguientes cinco atributos básicos (Nielsen 1994):

- Facilidad de aprendizaje: la facilidad de aprender la funcionalidad y comportamiento del sistema. Define en cuánto tiempo un usuario, que nunca ha visto una interfaz, puede aprender a usarla bien y realizar operaciones básicas.
- Eficiencia de uso: involucra alcanzar el nivel de productividad requerido, una vez que el usuario ha aprendido a usar el sistema. Determina la rapidez con que se pueden desarrollar las tareas.
- Retención sobre el tiempo: cuando un usuario ha utilizado un sistema tiempo atrás, y tiene la necesidad de utilizarlo de nuevo, la curva de aprendizaje debe ser significativamente menor que el caso del usuario que nunca haya utilizado dicho sistema. Esto es de primordial importancia para aplicaciones usadas intermitentemente.
- Tasa de error: la capacidad del sistema para ofrecer una tasa baja de errores, apoyar a los usuarios a cometer pocos errores durante el uso del sistema, y en caso de que cometan errores, ayudarles a recuperarse fácilmente.
- Satisfacción: se refiere a la impresión subjetiva del usuario respecto al sistema.

Métodos de evaluación de la usabilidad

Actualmente existe una variedad de métodos utilizados para evaluar la usabilidad. Algunos de estos métodos hacen uso de datos recolectados de usuarios y sus preferencias, mientras que otros confían en los expertos en usabilidad. A continuación, se listan varios de estos métodos.

1. Métodos de inspección: se basa en la disponibilidad de evaluadores que examinan si una interfaz determinada cumple una serie de principios de usabilidad. Estos métodos dependen de las opiniones, juicios e informes generados por los evaluadores de usabilidad (Sánchez 2011).
2. Análisis de tareas: el análisis de tareas significa aprender acerca de los objetivos y formas de trabajar de los usuarios. El análisis de tareas también puede significar averiguar qué tareas más específicas deben hacer los usuarios para cumplir con esos objetivos y qué medidas deben tomar para cumplir con esas tareas (Sánchez 2011).
3. Grupos de enfoque: un grupo de enfoque es un debate donde un moderador dirige a un grupo de participantes mediante una serie de preguntas sobre un tema en particular. Aunque suele utilizarse como una herramienta de marketing, a veces se utiliza para evaluar la usabilidad. Aplicada en la etapa de definición del producto, un grupo de 6 a 10 usuarios son reunidos para discutir lo que desean del producto. Un facilitador experimentado en grupos de enfoque es contratado para guiar la discusión a las áreas de interés de los desarrolladores. En general, los datos recogidos no son cuantitativos, pero pueden apoyar la obtención de una idea de la opinión del grupo de enfoque (Sánchez 2011).
4. Test de Usabilidad: los test de usabilidad son la práctica de usabilidad más extendida. Consisten en presentar al usuario una serie de tareas a realizar, y pedirle que las realice con el sistema. Las acciones y comentarios de usuario se recopilan para un análisis posterior. Para conseguir unos test de usabilidad con resultados fiables, las condiciones del test y del lugar donde éste se realiza deben ser lo más parecidas posibles al entorno de uso previsto para el sistema (Grau 2014).
5. Evaluación heurística: la evaluación heurística es un método de la ingeniería de la usabilidad, utilizado para encontrar y evaluar problemas de usabilidad en el diseño de las interfaces de usuario, como parte de un proceso de diseño iterativo. Involucra tener un conjunto de evaluadores expertos examinando las interfaces y utilizando los principios reconocidos de la usabilidad (las heurísticas) para categorizar y valorar los problemas descubiertos. Es el más popular de los métodos de inspección de la usabilidad, ya que es rápido, barato y fácil. Se desarrolló para ayudar en el diseño de las interfaces de usuario; confía en las revisiones de expertos para descubrir los problemas de usabilidad y es ampliamente utilizada debido a su

velocidad y la relación costo-efectividad. La lista de diez heurísticas de Jakob Nielsen es la más comúnmente utilizada en la industria; son diez principios generales para el diseño de interfaces de usuario. Son llamadas “heurísticas” debido a que tienen más naturaleza de reglas de oro, que directrices específicas de usabilidad. A continuación dichas heurísticas (Nielsen 1995):

5.1. Visibilidad del estado del sistema: el sistema debe siempre mantener a los usuarios informados del estado del sistema, con una retroalimentación apropiada y en un tiempo razonable.

5.2. Correspondencia entre el sistema y el mundo real: el sistema debe hablar el lenguaje de los usuarios, con las palabras, las frases y los conceptos familiares a ellos, en lugar de los términos orientados al sistema. Utilizar convenciones del mundo real, haciendo que la información aparezca en un orden natural y lógico.

5.3. Control y libertad para el usuario: los usuarios frecuentemente eligen funciones del sistema por error y necesitan una salida de emergencia claramente marcada, es decir, salir del estado indeseado sin tener que pasar por un diálogo extendido. Es importante disponer de opciones para deshacer y rehacer una acción.

5.4. Consistencia y estándares: los usuarios no deben tener que preguntarse si las diversas palabras, situaciones, o acciones significan la misma cosa. En general se deben seguir las normas y convenciones de la plataforma sobre la que se está implementando el sistema.

5.5. Prevención de errores: mejor que tener buenos mensajes de errores es el hecho de diseñar cuidadosamente las interfaces.

5.6. Reconocer más que recordar: minimizar la necesidad de memorizar al usuario, haciendo visibles objetos, acciones y opciones. El usuario no debería tener que recordar la información de una parte de diálogo de una interfaz a la otra. Instrucciones para el uso del sistema, deberían ser visibles o fácilmente encontradas en cualquier momento.

5.7. Flexibilidad y eficiencia de uso: los aceleradores no vistos por el usuario principiante, mejoran la interacción para el usuario experto de tal manera que el sistema puede servir para usuarios inexpertos o experimentados. Es importante que el sistema permita personalizar acciones frecuentes.

5.8. Un diseño estético y minimalista: no debe incluirse información que no sea aplicable o que raramente se necesite. Cada unidad adicional de la información en un diálogo compite con las unidades relevantes de la información y disminuye su visibilidad relativa.

5.9. Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de los errores: los mensajes de error se deben expresar en un lenguaje claro (sin códigos), indicar exactamente el problema y constructivamente sugerir una solución.

5.10. Ayuda y documentación: Aunque es mejor que el sistema pueda usarse sin documentación, puede ser necesario disponer de ella y adicionalmente de ayuda. Ésta ha de ser fácil de buscar, centrada en las tareas del usuario, tener información de las etapas a realizar y que no sea muy extensa.

Para la validación de esta variable se utilizó el método de evaluación heurística mediante la lista de chequeo (Ver anexo 2). Este método arrojó resultados satisfactorios, teniéndose en cuenta un antes y un después, donde se evidencia la contribución de la presente investigación, en el incremento del porciento de usabilidad del sistema, el cual ascendió de un 70.6% a un 88.2%, esto se calcula mediante la cantidad de indicadores satisfactorios con respecto al total de indicadores.

3.6. Conclusiones parciales

Al concluir el presente capítulo se puede afirmar que:

- Las métricas de validación del diseño permitieron definir la calidad del componente de gestión de plantillas de documentos de procesos en el SIGEF.
- Los resultados de la realización de pruebas de caja blanca, mediante la técnica del camino básico para la validación del código, reflejan que las operaciones internas se ajustan a las especificaciones y que los componentes internos funcionan de forma adecuada.
- La aplicación de las 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.
- Las pruebas de liberación evidenciaron que el componente cumple con las necesidades requeridas para ser entregado a la Fiscalía General de la República y facilita el trabajo en el proceso de atención al ciudadano.
- Se verificó la contribución de la presente investigación a la usabilidad del sistema, mediante la lista de chequeo.

CONCLUSIONES GENERALES

Luego de realizar el presente trabajo se concluye que:

- A partir del estudio realizado durante la investigación, se analizaron un grupo de elementos conceptuales que permitieron a los autores apropiarse de los conocimientos necesarios en el área de sistemas de gestión de información jurídica, lo cual propició elaborar la fundamentación teórica del presente trabajo.
- El análisis y diseño de la propuesta de solución permitió sentar las bases para la implementación de la solución propuesta.
- Se cumplió el objetivo de la investigación, pues se realizó la implementación del componente para la gestión de plantillas de documentos jurídicos en el Sistema de Gestión Fiscal, cumpliendo con todos los requisitos establecidos.
- Las métricas y pruebas utilizadas para validar el componente permitieron la obtención de un producto entregable con una calidad aceptable para mejorar el trabajo de los fiscales y contribuir a la usabilidad del sistema.

RECOMENDACIONES

Los objetivos de esta investigación fueron logrados satisfactoriamente, sin embargo, es necesario tener en cuenta la siguiente recomendación:

- Extender el uso del componente para gestión de plantillas de documentos en el Sistema de Gestión Fiscal a cada proceso llevado a cabo en la fiscalía.

REFERENCIAS BIBLIOGRÁFICAS

ACOSTA, J.C.P. y MACHADO, Y.F., 2016. *Sistema de Gestión Fiscal*. 2016. S.l.: s.n.

AGUILAR, L.J., 2003. *Fundamentos de programación: algoritmos y estructura de datos y objetos*. S.l.: s.n. ISBN 84-481-3664-0.

ÁLVAREZ, 2012. Manejando las vistas con Twig en Symfony2. *Maestros del Web* [en línea]. [Consulta: 28 febrero 2018]. Disponible en: http://www.maestrosdelweb.com/curso-symfony2-la-vista-twig/.

ÁLVAREZ, P., 2002. Sistema de Apoyo a los Fiscales, herramienta clave de la Reforma Procesal Penal en Chile. *SONDA | Líder Latinoamericano de Servicios de TI* [en línea]. [Consulta: 4 marzo 2018]. Disponible en: https://www.sonda.com/es/caso/gobierno/sistema-de-apoyo-a-los-fiscales-herramienta-clave-.

EGEA GARCÍA, C. 2007 *Diseño web para todos I* Icaria Editorial. Barcelona.

ALZINA, R.B., 2004. *Metodología de la investigación educativa*. S.l.: Editorial La Muralla. ISBN 84-7133-748-7.

AMBLER, S., 2005. *Enterprise unified process, the: extending the rational unified process*. S.l.: Prentice Hall Press. ISBN 0-13-191451-0.

ARMAND, S., 2014. *Extending Symfony2 Web Application Framework*. S.l.: Packt Publishing Ltd. ISBN 978-1-78328-720-8.

BOUCHAREINE, P., 2014. PostgreSQL: Documentation: 9.4: Release 9.4. [en línea]. [Consulta: 28 febrero 2018]. Disponible en: https://www.postgresql.org/docs/9.4/static/release-9-4.html.

CHIDAMBER, S.R. y KEMERER, C.F., 1994. A metrics suite for object-oriented design. *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476-493.

COBO, Á., 2005. *PHP y MySQL: Tecnología para el desarrollo de aplicaciones web.* S.l.: Ediciones Díaz de Santos. ISBN 84-7978-706-6.

COSTA, H.R.C., 2004. Estándares básicos de programación. *Scribd* [en línea]. [Consulta: 18 marzo 2018]. Disponible en: https://es.scribd.com/document/36991779/Estandares-basicos-de-programacion.

CUERVO, J., 2015. Revista Informática Jurídica. *Informática Jurídica* [en línea]. [Consulta: 13 marzo 2018]. Disponible en: http://www.informatica-juridica.com/revista/.

FISCALÍA GENERAL DE LA REPÚBLICA DE CUBA, 2015. La Fiscalía y su labor de formación vocacional. [en línea]. [Consulta: 22 febrero 2018]. Disponible en: http://www.fgr.cu/es/la-fiscalia-y-su-labor-de-formacion-vocacional.

FLANAGAN, D., 2006. *JavaScript: The Definitive Guide*. S.l.: O'Reilly Media, Inc. ISBN 978-0-596-10199-2.

REFERENCIAS BIBLIOGRÁFICAS

FOWLER, M., 2004. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. S.l.: Addison-Wesley Professional. ISBN 978-0-321-19368-1.

GAMMA, E., HELM, R., JOHNSON, R. y VLISSIDES, J., 1994. *Design patterns: elements of*. S.l.: Addison-Wesley.

GAUCHAT, J.D., 2012. *El gran libro de HTML5, CSS3 y Javascript*. S.l.: Marcombo. ISBN 978-84-267-1782-5.

GÓMEZ, O.T., LÓPEZ, P.P.R. y BACALLA, J.S., 2010. Criterios de selección de metodologías de desarrollo de software. *Industrial Data*, vol. 13, no. 2, pp. 70-74. ISSN 1560-9146, 1810-9993.

GONZÁLEZ, Y.D., 2012. Patrón Modelo-Vista-Controlador. *Revista Telem@ tica*, vol. 11, no. 1, pp. 47-57.

GRAU, X.F., 2014. *Principios Básicos de Usabilidad para Ingenieros de Software*. 2014. S.l.: s.n.

Guia_Breve_ISO690-2010.pdf, [sin fecha]. S.l.: s.n.

HAMON, H., 2014. Los patrones de diseño de los componentes Symfony. *symfony.es* [en línea]. [Consulta: 6 marzo 2018]. Disponible en: http://symfony.es/noticias/2014/05/03/los-patrones-de-diseno-de-los-componentes-symfony/.

IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 1990. pp. 1-84. DOI 10.1109/IEEESTD.1990.101064.

ISO, I., 1998. ISO 9241-11: 1998: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs)-Part 11: Guidance on Usability. *International Organization for Standardization. Retrieved from http://books. google. com. co/books*,

JOSÉ ROMÁN MANZ, 2009. ¿Qué es CSS? | CSS en español. [en línea]. [Consulta: 28 febrero 2018]. Disponible en: https://lenguajecss.com/p/css/introduccion/que-es-css.

LARMAN, C., 2003. *UML y Patrones*. S.l.: Pearson Educación ^ eMadrid Madrid. ISBN 1-5129-3650-2.

LARMAN, C., 2004. *Agile and Iterative Development: A Manager's Guide*. S.l.: Addison-Wesley Professional. ISBN 978-0-13-111155-4.

LELARGE, G., 2008. PgAdmin III - Guía Ubuntu. [en línea]. [Consulta: 28 febrero 2018]. Disponible en: https://www.guia-ubuntu.com/index.php?title=PgAdmin_III.

Lex-Doctor. *Softonic* [en línea], 2010. [Consulta: 4 marzo 2018]. Disponible en: https://lex-doctor.softonic.com.

LOERA, J., 2014. ¿De dónde proviene el CamelCase? - Aprende a Programar - Codejobs. [en línea]. [Consulta: 18 marzo 2018]. Disponible en: https://www.codejobs.biz/es/blog/2014/10/10/de-donde-proviene-el-camelcase.

REFERENCIAS BIBLIOGRÁFICAS

LORENZ, M. y KIDD, J., 1994. *Object-oriented software metrics: a practical guide*. S.l.: Prentice-Hall, Inc. ISBN 0-13-179292-X.

LOUDEN, K.C., 2008. *Lenguajes De Programación*. 2nd Revised edition edition. México: Cengage Learning Latin America. ISBN 978-970-686-284-6.

MARTÍNEZ, F. y MARTÍN, G., 2003. Introducción a la programación estructurada en C. *Universidad de Valencia*,

MARTÍNEZ, M.G., 2015. Características Técnicas del programa GEDEX Gestión de Expedientes Jurídicos. [en línea]. [Consulta: 4 marzo 2018]. Disponible en: https://www.brindys.com/es/caracteristicas-tecnicas.html.

MARTÍNEZ, Y.F., 2014. *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*. S.l.: Universidad de las Ciencias Informática.

MINISTERIO DE COMUNICACIONES, 2017. La informatización de la sociedad, una prioridad para Cuba | MINCOM. [en línea]. [Consulta: 13 marzo 2018]. Disponible en: http://www.mincom.gob.cu/?q=node/1434.

MONTOYA, C.E.G., 2013. Seguridad en la configuración del servidor web Apache. *INGE CUC*, vol. 9, no. 2, pp. 31-38. ISSN 2382-4700.

MORENO, B.V., 2015. Integridad de Datos: Definición y problemas. [en línea]. [Consulta: 25 mayo 2018]. Disponible en: https://www.tecnologias-informacion.com/integridaddatos.html.

NASER, A. y CONCHA, G., 2011. *El gobierno electrónico en la gestión pública*. S.l.: Cepal. ISBN 92-1-121767-9.

NIELSEN, J., 1994. *Usability engineering*. S.l.: Elsevier. ISBN 0-08-052029-4.

NIELSEN, J., 1995. 10 Heuristics for User Interface Design: Article by Jakob Nielsen. *Nielsen Norman Group* [en línea]. [Consulta: 27 mayo 2018]. Disponible en: https://www.nngroup.com/articles/ten-usability-heuristics/.

NIELSEN, J., 2012. Usability 101: Introduction to Usability. *Nielsen Norman Group* [en línea]. [Consulta: 27 mayo 2018]. Disponible en: https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

NORÉN, A., 2016. ¡Casos de prueba, que son, como se hacen y para qué sirven……! *Testing Colombia* [en línea]. [Consulta: 23 mayo 2018]. Disponible en: http://www.testingcolombia.com/casos-de-prueba-que-son-como-se-hacen-y-para-que-sirven/.

OSTERWALDER, A., 2005. Clarifying business models: Origins, present, and future of the concept. *Communications of the association for Information Systems*, vol. 16, no. 1, pp. 1.

OXFORD DICTIONARIES, 2015. retroacción | Definición de retroacción en español de Oxford Dictionaries. [en línea]. [Consulta: 18 marzo 2018]. Disponible en: https://es.oxforddictionaries.com/definicion/retroaccion.

REFERENCIAS BIBLIOGRÁFICAS

PÉREZ, J.E., 2015. CSS avanzado. [en línea]. [Consulta: 17 junio 2018]. Disponible en: http://librosweb.es/libro/css_avanzado/.

PRESSMAN, R.S., 2010. *Ingeniería del software - Un enfoque práctico - séptima edición*. 2010. S.l.: s.n.

SÁNCHEZ, T.R., 2015. *Metodología de desarrollo para la Actividad productiva de la UCI*. 2015. S.l.: s.n.

SÁNCHEZ, W., 2011. *La usabilidad en Ingeniería de Software: definición y características.* 2011. S.l.: s.n.

SIERRA, C.D., 2007. Visual Paradigm For Uml. [en línea]. S.l. [Consulta: 17 junio 2018]. Disponible en: https://es.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml.

SOMMERVILLE, I., 2005. *Ingeniería del software*. S.l.: Pearson Educación. ISBN 978-84-7829-074-1.

THOMSON, D.K., 2014. Software de Gestión Aranzadi Infolex. *Thomson Reuters* [en línea]. [Consulta: 4 marzo 2018]. Disponible en: https://www.thomsonreuters.es/content/spain/es/productos-servicios/aranzadi-infolex.html.

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS, 2002. Historia | Universidad de las Ciencias Informáticas. [en línea]. [Consulta: 22 febrero 2018]. Disponible en: http://www.uci.cu/universidad/historia.

VALDÉS, J.T., 1987. *Derecho informático*. S.l.: Universidad nacional autónoma de México.

VELASCO, R., 2016. Llega el nuevo NetBeans 8.2 con soporte completo para PHP7. *RedesZone* [en línea]. [Consulta: 28 febrero 2018]. Disponible en: https://www.redeszone.net/2016/10/03/llega-nuevo-netbeans-8-2-soporte-completo-php7/.

WAGE, J.H., 2016. Home — Doctrine Project. [en línea]. [Consulta: 19 marzo 2018]. Disponible en: http://www.doctrine-project.org/index.html.

ZAPATA, C.M., 2007. *Un método para el refinamiento interactivo del diagrama de clases UML* [en línea]. 2007. S.l.: s.n. Disponible en: http://www.scielo.org.co/pdf/dyna/v74n153/a27v74n153.pdf.