Universidad de las Ciencias Informáticas
Facultad 3

“Herramienta para la generación de casos de pruebas a partir de técnicas de pruebas funcionales”

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autores: Ricardo Longo Zamora

Ángel Rafael Álvarez Pérez

Tutor: Ing. Denia Madruga Hernández

La Habana, mayo de 2016.

“Año del 58 de la Revolución”

Calidad significa hacer las cosas bien incluso cuando nadie te está mirando.

Henry Ford.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de ________ del año ________.

"[Insertar nombre(s) de autor(es)]"	"[Insertar nombre(s) de tutor(es)]"
____________________________	___________________________

AGRADECIMIENTOS

Ricardo Longo Zamora

Mamá, Papá, Abuela, Abuelo, no hay manera de explicar lo agradecido que estoy de ustedes, para entenderlo, tienes que sentirlo. Gracias por los consejos y regaños que me han hecho ver el mundo de manera diferente, me han convertido en el yo que ven ante ustedes. Aunque a veces soy frío y de metal por fuera, quiero que sepan que por dentro soy vulnerable a su amor.
Agradezco a mi familia por apoyarme en todo momento, aunque estén lejos o cerca, presentes o no, les doy las gracias por creer en mí.
Ailén has entrado en un momento crucial en mi vida, y te has quedado en ella para siempre. Gracias por apoyarme en todo momento, sin ti hubiese sido difícil seguir el camino. TE AMO.
Gracias a mis amigos, que han estado presente en las buenas y las malas, gracias a Deyler por haberme llevado a conocer a Yoan, sin su ayudada hubiese llegado a este momento.
A mi compañero de tesis, que después que le halas una oreja hasta que no termina el trabajo no se acuesta a dormir, a mi tutora Denia y todas aquellas personas que me han dedicado parte de su tiempo para ayudarme en este trabajo. GRACIAS A TODOS.

Ángel Rafael Álvarez Pérez

Mamá, no hay palabras para describir cuanto agradezco tu apoyo incondicional, gracias por haber estado ahí siempre para mí, por tus conejos, tu regaños, gracias por existir.
Papá, gracias por apoyarme incluso cuando no estábamos de acuerdo, la persona que soy hoy es a tus enseñanzas.
A mis tíos Sergio, Rosaura, Raúl, Pedro gracias por todo su apoyo, yo he tenido la suerte de tener muchos padres y eso es porque ustedes existen.
Mi abuela Nena y mi abuelo Antonio, mis viejitos, gracias por malcriarme, por su amor, sus cuidados, a Uds. debo mi vida y les estaré siempre agradecido.
A Alioska, mi viejo, llegaste a mi vida y sin preguntar ni cuestionar me diste tu amor y me llamaste hijo, tu único defecto siempre ha sido y será ser demasiado bueno conmigo, pero por eso te lo voy a agradecer siempre.
A Giannis, por tu cariño, por cuidar tan bien de mi papá y por defenderme siempre y estar de mi lado cada vez que le llevaba la contraria al viejuco gracias,
A Adri y Arley mis hermanos, estos han sido los agradecimientos más difíciles que eh escrito, no tengo palabras para agradecerle el haber estado estos 5 años a mi lado aguantando mis pesadeces, mi mal genio y los otros muchos de mis defectos, pero a pesar de esto nunca me fallaron y siempre que resbalé ahí estuvieron para sostenerme.
David, no solo la sala es tu cuarto ni el teléfono es tuyo, mi casa es tu casa y todo lo que tengo es tuyo, gracias por regañarme, alarme las orejas, por dejar de dormir para repasarme para las pruebas. Es como dijo tu papá, la amistad que hemos forjado va a durar para siempre por eso gracias.
A mis amigos, los de la UCI y los de Sta. Clara, a todos los que pusieron al menos un grano de arena, a todos los que me ayudaron, a todos los que me apoyaron. A mi compañero de tesis que es un bestia (en el buen sentido de la palabra) y no hubiera querido otro, a mi tutora. A todos GRACIAS.

DEDICATORIA

Yo Ricardo Longo Zamora dedico este trabajo a:

Mis padres y mis abuelos por todo el apoyo que me han dado durante el transcurso de estos cinco años y de mi vida.

Yo Ángel Rafael Álvarez Pérez dedico este trabajo a:

Mi abuelo Antonio, por ser mi faro en las más oscuras tormentas, y modelo de hombre en el cual quiero convertirme.

A mi abuelo Manuel Antonio por ser la persona más maravillosa que he conocido, aunque ya no estés conmigo sé que desde el cielo aún me cuidas.

A mi mamá y a mi papa por darme la vida y darme todo lo que estuvo a su alcance.

RESUMEN

El presente trabajo, titulado "Herramienta para la generación de casos de pruebas a partir de técnicas de pruebas funcionales" está enfocado a la mejora de la calidad del proceso de pruebas en los proyectos de la Facultad 3 específicamente el Centro de Gobierno Electrónico (CEGEL) de la Universidad de Ciencias Informáticas (UCI), teniendo en cuenta que en ocasiones existe pérdida de información y se introducen errores producto al extenso volumen de información y que los casos de pruebas son realizados por diferentes analistas de forma manual.

Para ello fue necesario realizar una fundamentación teórica en la que se abordan los diferentes conceptos emitidos por varios autores referentes a aspectos relacionados con el Proceso de Desarrollo de Software, Calidad de Software, Pruebas de Software, entre otros, que sustentan el hilo conductor del marco teórico referencial. Por otra parte el desarrollo de la herramienta estuvo guiado por las especificaciones que propone la metodología AUP-UCI, llevando a cabo la implementación de la propuesta de solución.

Con el desarrollo de esta herramienta web y su incorporación en los proyectos se pretende minimizar la carga de trabajo que realizan los analistas de los equipos de desarrollo de software. Además se contará con una base de datos centralizada para todos los proyectos, donde se guardará toda la información referida a los casos de prueba así como los documentos de especificación de requisitos.

PALABRAS CLAVE

Proceso de Desarrollo de Software, Calidad de Software, Pruebas de Software

Índice de Contenido

AGRADECIMIENTOS I

DEDICATORIA II

RESUMEN III

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 6

1. Introducción 6
1.1. Proceso de desarrollo de software 6
1.2. Tendencias en el desarrollo de software 10
1.3. Uso de aplicaciones web 10
1.3.1. Marcos de trabajo 11
1.3.2. Patrón Modelo Vista Controlador 13
1.4. Metodologías de desarrollo de software 14
1.5. Conclusiones parciales 15

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DEL SISTEMA 17

2. Introducción 17
2.1. Metodología de desarrollo de software 17
Metodología AUP-UCI 17
2.2. Fase de Inicio 18
2.2.1. Calidad de software 18
2.2.2. Pruebas funcionales de software 19
2.2.3. Diseño de casos de prueba 20
2.2.4. Técnicas de pruebas funcionales 20
2.2.5. Partición de equivalencia 21
2.2.6. Tabla de decisión 21
2.2.7. Generación de casos de pruebas funcionales 22
2.3. Fase de Ejecución 23
2.3.1. Historia de usuario 23

2.3.2. Requisitos funcionales y no funcionales 25
2.3.2. Validación de requisitos 27
2.4. Descripción de la propuesta de solución 28
2.4.1. Prototipos 28
2.5. Selección de tecnologías 29
2.6. Diagrama de clases del diseño 32
2.6.1. Diseño arquitectónico 34
2.7. Implementación 42
2.7.1. Estándares de codificación 42
2.8. Conclusiones parciales 43
CAPÍTULO 3: VERIFICACIÓN Y VALIDACIÓN 44
3. Introducción 44
3.1. Fase de cierre 44
3.2. Métricas 44
3.2.1. Métricas para validar el diseño 44
3.3. Estrategia de pruebas 51
3.4. Verificación de la propuesta 51
3.5. Validación de la propuesta 55
3.6. Conclusiones parciales 57
CONCLUSIONES 59
RECOMENDACIONES 60
BIBLIOGRAFÍA 61

Índice de Figuras

FIGURA 1: PATRÓN ARQUITECTÓNICO MODELO VISTA CONTROLADOR. 13

FIGURA 2: PROCESO PARA LA GENERACIÓN DE CASOS DE PRUEBAS FUNCIONALES (PALACIO, 2009). 23

FIGURA 3: CREAR CASO DE PRUEBA CON PARTICIÓN DE EQUIVALENCIA. 28

FIGURA 4: CREAR CASO DE PRUEBA CON TABLA DE DECISIÓN. 29

FIGURA 5: DIAGRAMA DE CLASES DEL DISEÑO. 33

FIGURA 6: ESQUEMA DE LA ARQUITECTURA DEL SISTEMA. 35

FIGURA 7: DIAGRAMA MVC. 36

FIGURA 8: PATRÓN EXPERTO. 38

FIGURA 9: PATRÓN ALTA COHESIÓN. 39

FIGURA 10: PATRÓN BAJO ACOPLAMIENTO. 39

FIGURA 11: PATRÓN CONTROLADOR. 40

FIGURA 12: MODELO DE DATOS RELACIONAL. 41

FIGURA 13: REPRESENTACIÓN EN PORCIENTO DE LOS RESULTADOS OBTENIDOS, OBTENIDOS TRAS APLICAR LA MÉTRICA TOC. 46

FIGURA 14: VALOR DEL ATRIBUTO DE CALIDAD RESPONSABILIDAD. 46

FIGURA 15: VALOR DEL ATRIBUTO DE CALIDAD COMPLEJIDAD DE IMPLEMENTACIÓN. . 47

FIGURA 16: VALOR DEL ATRIBUTO DE CALIDAD REUTILIZACIÓN. 47

FIGURA 17: REPRESENTACIÓN EN PORCIENTO DE LA APLICACIÓN DE LA MÉTRICA RC EN INTERVALOS DEFINIDOS. 49

FIGURA 18: GRADO DE AFECTACIÓN DEL ATRIBUTO ACOPLAMIENTO. 49

FIGURA 19: GRADO DE AFECTACIÓN DEL ATRIBUTO COMPLEJIDAD DE MANTENIMIENTO. 50

FIGURA 20: GRADO DE AFECTACIÓN DEL ATRIBUTO CANTIDAD DE PRUEBAS. 50
FIGURA 21: CÓDIGO FUENTE QUE GENERA LA FUNCIONALIDAD MODIFICAR MÓDULO. 52
FIGURA 22: GRAFO DE FLUJO. 53
FIGURA 23: CANTIDAD DE NC. 56
FIGURA 24: TIEMPO DE DEMORA. 57

Índice de Tablas

TABLA 1: HU 1: CREAR CASO DE PRUEBA (PARTICIÓN DE EQUIVALENCIA). 24

TABLA 2: HU 2: CREAR CASO DE PRUEBA (TABLA DE DECISIÓN). 24

TABLA 3: RELACIÓN DE NÚMEROS DE PROCEDIMIENTO POR CLASE. 45

TABLA 4: RELACIONES DE USO ENTRE CLASES.. 48

TABLA 5: CAMINOS INDEPENDIENTES ESTABLECIDOS. 53

TABLA 6: NO CONFORMIDADES DETECTADAS EN LAS PRUEBAS DE LIBERACIÓN. 55

INTRODUCCIÓN

El éxito de la industria del software está potenciado por el grado de innovación y calidad de sus productos, haciendo que las empresas dediquen mayor tiempo y presupuesto al desarrollo de estos. La calidad es un factor clave a tener en cuenta, dicho aspecto es motivo de preocupación entre los productores de software en la creación de sus productos para que sean fiables, seguros y funcionalmente operativos.

Las pruebas de software son un conjunto de pasos donde se pueden incluir técnicas y métodos para la verificación y validación, son la ejecución de un programa con el fin de detectar errores. Son un proceso que se divide en subprocesos, los cuales son, la planificación de las pruebas, con vista a, dónde, con qué, y con quién se va a realizar las pruebas. El diseño de las pruebas, está dado por el diseño de los elementos que van a permitir ejecutar la prueba. La ejecución de las pruebas, no es más que ejecutar las pruebas diseñadas con anterioridad, y la evaluación de los resultados, viene dada por la resolución de no conformidades que se van a detectar con estas pruebas. La presente investigación se enfoca en el diseño de casos de pruebas que facilitará establecer el posible comportamiento del sistema ante una serie de valores determinados basándose en la creación de casos de prueba cuya ejecución permitirá el aumento de detección de errores (Pressman, 2005).

Debido al vertiginoso desarrollo actual de las Tecnologías de la Información y las Comunicaciones, Cuba tiene un gran interés en adentrarse en el mercado del software, llevando a cabo la inmensa labor para informatizar el país en búsqueda de los beneficios para la sociedad. La UCI se destaca como una institución abanderada donde se produce gran parte del software dedicado a la informatización del país. La UCI tiene como objetivo formar profesionales en las ciencias informáticas, capacitando mejor a sus estudiantes con la vinculación de los mismos en entornos productivos reales desde los primeros años de la carrera.

CEGEL, es un centro de desarrollo de software con varios proyectos y a cada uno de estos se le efectúa el proceso de pruebas funcionales, generando casos de pruebas, durante su ciclo de vida. Además dentro del centro se ha creado un grupo de calidad para realizar pruebas funcionales a los proyectos que allí confinan. Los casos de pruebas son generados manualmente por los analistas del centro, donde estos, analizan todas las posibles operaciones que pueda realizarse con el producto de software, donde se tienen en cuenta las variables y relaciones entre las funcionalidades. En ocasiones existe pérdida de información y se introducen errores debido al extenso volumen de información y que los casos de pruebas son realizados por diferentes

analistas de forma manual. Los casos de pruebas realizados en el centro CEGEL son incorrectos y se ejecutan pruebas deficientes, gastos innecesarios de recursos y no se detectan la totalidad de errores posibles. A causa de la no completa detección de errores, producto a casos de pruebas deficientes realizados con antelación, trae como consecuencias una disminución de la calidad del producto y la insatisfacción de los clientes al considerar que el software no cumple los requisitos de calidad para los que fue creado, por lo tanto, no cumple con el propósito que inicialmente se debatió con el cliente respecto a su producto.

A la hora de realizar las pruebas se debe tener muy claro qué se quiere probar, utilizando para ello los tipos de pruebas, por ejemplo las funcionales. Para generar los casos de pruebas se utilizan diferentes métodos de prueba caja blanca o caja negra. El primero prueba las funcionalidades del sistema a partir del código verificando que es ejecutado totalmente. Mientras que el segundo método lo hace utilizando las interfaces de usuario sin importarle la implementación. Estos métodos se complementan entre sí, aumentando la probabilidad de detectar errores con los casos de pruebas. Cada método de prueba define un conjunto de técnicas para cumplir el objetivo de prueba, ejemplo, Partición de Equivalencia y Tabla de Decisión, encaminadas a realizar casos de pruebas para unidades específicas, requisitos o casos de uso y otras lo que hacen es relacionar grupos de requisitos utilizados en la presente investigación. El diseño de casos de prueba es un proceso costoso, porque requiere la presencia de varias personas y gran cúmulo de tiempo y dedicación. Además, es complejo, pues requiere el conocimiento profundo del sistema a probar y su comportamiento ante diversas entradas, que es la generación de casos de pruebas de forma manual (Heumann, 2001).

Teniendo en cuenta lo anteriormente expuesto se define como **problema a resolver** de la presente investigación ¿Cómo obtener casos de pruebas basados en técnicas de pruebas funcionales que permitan disminuir el tiempo y las no conformidades asociadas a su diseño en el Centro de Gobierno Electrónico?

El **objeto de estudio** de la investigación se centra en el desarrollo de software y se concreta como **campo de acción** el proceso de desarrollo de software.

Se define como **objetivo general** desarrollar una herramienta para la generación de casos de prueba, a partir del documento de especificación de requisitos, de manera que contribuya a disminuir el tiempo necesario para la generación de los casos de pruebas del centro CEGEL.

Para dar cumplimiento al objetivo general del trabajo de diploma en cuestión se identificaron los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación mediante un estudio de los referentes teóricos que guían actualmente el proceso de desarrollo de software, identificando buenas prácticas para la construcción de una aplicación con un enfoque de calidad.
- Caracterizar el proceso de generación de casos de pruebas basados en técnicas de pruebas funcionales, los principales conceptos asociados a la generación de casos de pruebas funcionales de software y las metodologías de desarrollo, técnicas y herramientas.
- Implementar la solución propuesta, mediante las herramientas y tecnologías seleccionadas.
- Validar la solución propuesta.

Para dar cumplimiento a los objetivos específicos se trazaron las siguientes **tareas de la investigación:**

- Estudio de los referentes teóricos del proceso de desarrollo de software.
- Análisis de las técnicas de pruebas seleccionadas.
- Identificación de metadatos en la especificación de requisitos de software para la ejecución de la técnica de prueba.
- Selección de la metodología y herramientas a utilizar para el desarrollo del sistema.
- Determinación de los requisitos funcionales y no funcionales de la herramienta a partir de las tendencias actuales de las pruebas de software.
- Diseño de la propuesta de solución garantizando alta cohesión y bajo acoplamiento entre clases.
- Codificación de la propuesta de solución poniendo en práctica las tendencias en el desarrollo de software.
- Generación de documentos de casos de pruebas usando la Técnica de Prueba Funcional Tabla de Decisión.

- Generación de documentos de casos de pruebas usando la Técnica de Prueba Funcional Partición Equivalencia.

Para dar respuesta a las tareas planteadas con anterioridad se proponen **Métodos Científicos** que se clasifican en:

Teóricos:

- **Análisis Histórico-Lógico:** es utilizado para realizar un estudio exhaustivo sobre el objeto de estudio, facilitando el análisis y comprensión del tema.
- **Analítico-Sintético:** este método se utiliza para analizar y comprender la teoría y documentación relacionada con el tema de investigación, permitiendo extraer los elementos más afines e importantes relacionados con el objeto de estudio.
- **Modelación:** se utiliza para el análisis y diseño del comportamiento de la aplicación, mediante el cual se pueden crear abstracciones con el propósito de explicar la realidad y permita además la codificación del sistema.

Empíricos:

- **Medición:** este método se emplea para evaluar los resultados obtenidos en las pruebas realizadas al software.

A continuación, se describe la estructura de los capítulos que conforman la investigación para facilitar el entendimiento de los lectores:

Capítulo 1. Fundamentación teórica

Este capítulo aborda los aspectos teóricos que rigen el desarrollo de la investigación. Se selecciona un modelo que relacione el objeto de estudio y el campo de acción de la investigación. Se analizan las tendencias actuales en el mundo del software y que guiaran el desarrollo de la solución.

Capítulo 2. Análisis, diseño e implementación de la propuesta de solución

Este capítulo se centra en la aplicación de la metodología de desarrollo seleccionada para obtener la solución a partir de la implementación de sus fases y disciplinas.

Capítulo 3. Verificación y Validación

Este capítulo es donde se aplican las pruebas usadas para validar la solución y se realizan los análisis necesarios para garantizar el cumplimiento de las variables planteadas en el problema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1. Introducción

El presente capítulo realiza el análisis de los contenidos teóricos que sustentan la investigación, orientados al proceso de desarrollo de software, permitiendo la creación de un marco conceptual que sustente la solución, centrado en las tendencias actuales de desarrollo de software.

1.1. Proceso de desarrollo de software

El software es el producto creado y mantenido por los ingenieros de software, estos incluyen todos los programas que se ejecutan dentro de un equipo de cómputo, independiente de su tamaño y arquitectura, el contenido que se presenta conforme a los programas se ejecutan y aquellos documentos físicos y virtuales que engloban todas las formas de medios electrónicos (Pressman, 2005).

Cada vez con mayor frecuencia, los individuos y la sociedad se apoyan en los avanzados sistemas de software. Por ende, se requiere producir económica y rápidamente sistemas confiables. A menudo resulta más barato a largo plazo usar métodos y técnicas de ingeniería de software para los sistemas de software, que sólo diseñar los programas como si fuera un proyecto de programación personal. Para muchos tipos de software, la mayoría de los costos consisten en cambiar el software después de ponerlo en operación (Sommerville, 2011).

El papel del software informático ha sufrido un cambio significativo durante un periodo de tiempo superior a 50 años. Enormes mejoras en rendimiento del hardware, profundos cambios de arquitecturas informáticas, grandes aumentos de memoria y capacidad de almacenamiento y una gran variedad de opciones de entrada y salida han conducido a sistemas más sofisticados y más complejos basados en computadoras (Pressman, 2005).

Al comienzo de los años 90. El escritor y futurista Alvin Toffler describió un cambio de poder en el que las viejas estructuras de poder (gubernamentales, educativas, industriales, económicas y militares) se desintegrarían a medida que las computadoras y el software nos llevaran a la democratización del conocimiento (Pressman, 2005).

Hoy en día, la computación omnipresente ha producido una generación de aplicaciones de información que tienen conexión en banda ancha a la web para proporcionar una capa de conexión sobre nuestras casas, oficinas y autopistas. El papel del software continúa su expansión (Pressman, 2005).

Cuando se trabaja para construir un producto de software es importante una serie de pasos, pautas que ayuden a crear un resultado de alta calidad y a tiempo, estas pautas es lo que se conoce como desarrollo de software. De acuerdo a Roger S. Pressman la definición exacta de este proceso sería *"un marco de trabajo para las tareas que se requieren en la construcción de software de alta calidad"* (Pressman, 2005).

Sommerville en consonancia con el concepto anteriormente mencionado hace alusión al proceso de desarrollo de software como una *"secuencia de actividades que conducen a la elaboración de un producto de software"*.

De acuerdo con los autores se infiere como concepto final que el proceso de desarrollo de software es la secuencia de actividades requeridas que conducen a la elaboración de un producto de software de alta calidad.

En ocasiones es complejo definir categorías genéricas para las aplicaciones del software, acorde aumenta la complejidad del software, es más difícil establecer comportamientos nítidamente separados. (Pressman, 2005). A continuación se muestran las clasificaciones definidas para el software:

Software de sistema:

La definición dada por Pressman yace en que es un conjunto de programas que han sido escritos para servir a otros programas. Algunos programas de sistemas (por ejemplo: compiladores, editores y utilidades de gestión de archivos) procesan estructuras de información complejas pero determinadas. Otras aplicaciones de sistemas (por ejemplo: ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones) procesan datos en gran medida indeterminados. En cualquier caso, el área del software de sistemas se caracteriza por una fuerte interacción con el hardware de la computadora; una gran utilización por múltiples usuarios; una operación concurrente que requiere una planificación, una compartición de recursos y una sofisticada gestión de procesos; unas estructuras de datos complejas y múltiples interfaces extremas (Pressman, 2005).

Aguilera propone la siguiente definición: está formado por todos aquellos programas cuya finalidad es servir al desarrollo o al funcionamiento de otros programas. Estos programas son muy variados: editores, compiladores, sistemas operativos, entornos gráficos, programas de telecomunicaciones, etc. pero se caracterizan por estar muy próximos al hardware, por ser utilizados concurrentemente por numerosos usuarios y por tratarse de programas de amplia difusión, no estando diseñados normalmente a medida. Esto permite un mayor esfuerzo en su diseño y optimización, pero también les obliga a ser muy fiables,

cumpliendo estrictamente las especificaciones para las que fueron creados. Un ejemplo de este tipo de software son los sistemas operativos, como Windows y Unix (Aguilera, 2010).

Realizado un previo estudio de los autores, un software de sistema consiste en un software que sirve para controlar e interactuar con el sistema operativo, proporcionando control sobre el hardware y dando soporte a otros programas.

Software de tiempo real:

Pressman define el software de tiempo real como el software que coordina/analiza/controla sucesos del mundo real conforme ocurren, se denomina de tiempo real. Entre los elementos del software de tiempo real se incluyen: un componente de adquisición de datos que recolecta y da formato a la información recibida del entorno externo, un componente de análisis que transforma la información según lo requiera la aplicación, un componente de control/salida que responda al entorno externo, y un componente de monitorización que coordina todos los demás componentes, de forma que pueda mantenerse la repuesta en tiempo real (típicamente en el rango de un milisegundo a un segundo) (Pressman, 2005).

La definición dada por Sergio Aguilera parte de que está formado por todos aquellos programas que miden, analizan y controlan los sucesos del mundo real a medida que ocurren, debiendo reaccionar de forma correcta a los estímulos de entrada en un tiempo máximo prefijado. Deben, por tanto, cumplir unos requisitos temporales muy estrictos y, dado que los procesos que controlan pueden ser potencialmente peligrosos, tienen que ser fiables y tolerantes a fallos. Por otro lado, no suelen ser muy complejos y precisan de poca interacción con el usuario. Un sistema de tiempo real es aquel en el que para que las operaciones computacionales estén correctas no depende solo de que la lógica e implementación de los programas computacionales sea correcto, sino también en el tiempo en el que dicha operación entregó su resultado. Si las restricciones de tiempo no son respetadas el sistema se dice que ha fallado (Aguilera, 2010).

Respecto a los autores estudiados un software de tiempo real, es aquel que interacciona con su entorno físico y responde a los estímulos del entorno dentro de un plazo de tiempo determinado. No basta con que las acciones del sistema sean correctas, sino que, además, tienen que ejecutarse dentro de un intervalo de tiempo determinado y precisan poca interacción con el usuario.

Software de Ingeniería y científico:

Otro de los campos clásicos de aplicación de la informática. Se encarga de realizar complejos cálculos sobre datos numéricos de todo tipo. En este caso la corrección y exactitud de las operaciones que realizan es uno de los requisitos básicos que deben de cumplir.

Pressman señala que el software de ingeniería y científico está caracterizado por los algoritmos de manejo de números. Las aplicaciones van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricación automática. Sin embargo, las nuevas aplicaciones del área de ingeniería-ciencia se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (del inglés CAD), la simulación de sistemas y otras aplicaciones interactivas, han comenzado a coger características del software de tiempo real e incluso del software de sistemas (Pressman, 2005).

Por otra parte Aguilera expresa que el campo del software científico y de ingeniería se ha visto ampliado últimamente con el desarrollo de los sistemas de diseño, ingeniería y fabricación asistida por ordenador (CAD, CAE y CAM), los simuladores gráficos y otras aplicaciones interactivas que lo acercan más al software de tiempo real e incluso al software de sistemas (Aguilera, 2010).

Software de gestión:

El proceso de la información que constituye la mayor de las áreas de aplicación del software y los sistemas discretos ha evolucionado hacia el software de sistemas de información de gestión (SIG) que acceden a una o más bases de datos que contienen información. La aplicación de estos restructuran datos existentes para facilitar las operaciones y gestionar la toma de decisiones, además de las tareas convencionales de procesamientos de datos (Pressman, 2005).

Sergio Águila propone que el procesamiento de información de gestión constituye, casi desde los inicios de la informática la mayor de las áreas de aplicación de los ordenadores. Estos programas utilizan grandes cantidades de información almacenadas en bases de datos con el objetivo de facilitar las transacciones comerciales o la toma de decisiones. Además de las tareas convencionales de procesamiento de datos, en las que el tiempo de procesamiento no es crítico y los errores pueden ser corregidos a posteriori, incluyen programas interactivos que sirven de soporte a transacciones comerciales (Aguilera, 2010).

Resumiendo el análisis de los autores previo se define como software de gestión, que el procesamiento de la información de gestión constituye la mayor área de aplicaciones de software. Además, estos facilitan las tareas administrativas, como el manejo y conservación de información, por ejemplo, de proyectos, usuarios, documentos y almacenan grandes cantidades de información en bases de datos para facilitar las transacciones comerciales o la toma de decisiones y realizan cálculos iterativos.

El proceso de desarrollo de software es de gran importancia pues ofrece estabilidad y orden a una actividad que puede volverse caótica si no se controla. Sin embargo un enfoque de ingeniería debe ser ágil, requerir

solo aquellas actividades, controles y documentaciones acuerdo al equipo de desarrollo y el producto que ha de producirse. En el siguiente epígrafe se analiza la tendencia que persigue el proceso de desarrollo de software, con el objetivo de tener un enfoque en relación al software que se implementará para darle solución al problema planteado en la presente investigación.

1.2. Tendencias en el desarrollo de software

La cuestión de cómo el desarrollo de software debe ser organizado con el fin de ofrecer, mejor y más rápidas soluciones se ha discutido en los círculos de ingeniería de software por décadas. Muchas revisiones de mejora han sido sugeridas, a partir de la estandarización y la medición del proceso de software a una multitud de herramientas concretas, técnicas, y prácticas. Actualmente, muchas de las propuestas de mejora han venido de profesionales experimentados, que tienen etiquetados métodos de desarrollo de software ágiles. Este movimiento ha tenido un gran impacto en cómo se desarrolla el software en todo el mundo (Trondheim, 2011).

Con la llegada de Internet han surgido las aplicaciones web. Este tipo de aplicaciones permiten usar la infraestructura de la web para desarrollar aplicaciones globales. Reportes recientes indican que las aplicaciones web representan más de la mitad del total de todas las aplicaciones de la industria de software. De tal forma queda indicado que las aplicaciones web siguen creciendo y ganando popularidad en un mercado muy competitivo, y que se perfilan a ser las aplicaciones del futuro (Mora, 2011).

Continuamente el desarrollo de software enfocado a métodos ágiles se encuentra estrechamente relacionado con el entorno web producto a los rápidos cambios de la industria perfilados al creciente avance del mundo móvil. Las aplicaciones basadas en web son herramientas informáticas accesibles desde un navegador web, ostentando como gran ventaja, de no invertir en grandes máquinas, porque estas pueden estar instaladas en servidores de la red (Aguilera, 2010), con el objetivo de facilitar el acceso a empresas y usuarios de las mismas con respectivos privilegios. Por tanto, es necesario analizar el uso de las aplicaciones web para comprenderlas mejor y así obtener los conocimientos necesarios para darle solución al problema planteado en la investigación.

1.3. Uso de aplicaciones web

La tendencia actual para el proceso de desarrollo de software es el uso de las aplicaciones web, que se han convertido en una herramienta de alto peso para el desarrollo de las empresas. Estas aplicaciones

presentan una serie de ventajas con lo cual se logra aprovechar y acoplar los recursos de empresas de una forma mucho más práctica que el software tradicional.

Una aplicación web es un sistema informático que los usuarios utilizan sus servicios a través de la red, accediendo a un servidor web. Las aplicaciones web son populares debido a la practicidad de los navegadores web que actualmente están disponibles en equipos de escritorio y portátiles. La facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software en la gran cantidad de potenciales clientes es otra razón de su popularidad. Entre sus principales ventajas destacan las siguientes:

Multiplataforma: Una misma versión de la aplicación puede ejecutarse sin problemas en múltiples plataformas tales como Windows, Linux, Mac, Android, etc.

Actualización: Las aplicaciones web siempre se mantienen actualizadas y no requieren que el usuario deba descargar actualizaciones y realizar tareas de instalación.

Portabilidad: Acceso inmediato y desde cualquier lugar, las aplicaciones basadas en tecnologías web no necesitan ser descargadas, instaladas o configuradas. Son independientes del ordenador donde se utilicen, pueden ser accedidas desde cualquier computadora conectada a la red.

Menos requerimientos de hardware: Pueden funcionar en cualquier equipo que disponga de un navegador web. Esto aplica tanto a celulares, tabletas como computadoras u otros dispositivos modernos.

Seguridad en los datos: los datos se alojan en servidores ubicados en centros de datos con toda la infraestructura necesaria para asegurar la protección de datos y el funcionamiento constante de las aplicaciones.
Para agilizar el desarrollo de una aplicación web, se crean los marcos de trabajo, debido a que estos contienen grupos de funcionalidades prediseñadas y una previa arquitectura implementada.

1.3.1. Marcos de trabajo

Con el paso de los años y la constante evolución de las tecnologías con el objetivo de facilitar el desarrollo de aplicaciones web fueron creados los marcos de trabajo (frameworks). Estos frameworks brindan funcionalidades, clases, modelos, arquitecturas los cuales fueron previamente definidos.
El Marco de trabajo para el desarrollo de aplicaciones web es el responsable de proveer un entorno de ejecución para los proyectos con funcionalidades y características de los conocidos marcos de trabajo. Brinda prestaciones que aseguran un alto grado de calidad e integración de las aplicaciones realizadas en

dicha plataforma tecnológica. Aspectos como la seguridad, el monitoreo de trazas, la gestión del personal, los reportes, entre otros, son contenidos entre las funcionalidades de la solución. Por lo que permite la obtención de soluciones con un alto grado de robustez, dadas por el número de funcionalidades brindadas durante el desarrollo de soluciones web (Xetid, 2012).

Martínez describe al marco de trabajo como un subsistema expandible de un conjunto de servicios, es un conjunto cohesivo de interfaces y clases que colaboran para proporcionar los servicios de la parte central e invariable de un subsistema lógico (Martínez, 2013).

Beneficios:

Seguridad: Las aplicaciones cuentan con una mayor seguridad en cuanto a almacenamiento y conexiones a las bases de datos. Permite mejores tiempos de prevención y recuperación ante errores y situaciones excepcionales. Garantiza que cada proyecto se ejecute bajo las mismas tecnologías bases (Xetid, 2012).

Arquitectónicos: Provee mecanismos que permiten la obtención de soluciones escalables a gran nivel de forma ligera y sencilla. Promueve el desarrollo seguro, ágil y basado en componentes de aplicaciones web empresariales (múltiples propósitos, gran complejidad y grandes volúmenes de datos) (Xetid, 2012).

Uso: Es ágil y capaz de centrar el desarrollo en el negocio, los requerimientos y las interfaces de usuario. Minimiza la ejecución de software a la medida y aumenta la reutilización de código fuente. Logra minimizar los tiempos de desarrollo de los productos resultantes y la estandarización de los mismos promoviendo buenas prácticas de desarrollo tales como, el uso de patrones de forma estandarizada (Xetid, 2012).

Finalmente se concluye que un marco de trabajo es un conjunto de componentes de software prefabricado con el objetivo de extender, personificar y desarrollar soluciones informáticas. Los mismos le permiten al desarrollador implementar aplicaciones teniendo una base prediseñada, agilizando el trabajo. Los marcos de trabajo para las aplicaciones web, permiten el desarrollo de sitios web dinámicos y aplicaciones web. El propósito de estos es permitir a los desarrolladores centrarse en los aspectos interesantes, aliviando la típica tarea repetitiva asociada con patrones comunes de desarrollo web.

Hay una amplia gama de marcos de trabajo para aplicaciones web que son distribuidos bajo licencia Open Source (Código abierto) que utilizan el patrón de arquitectura de software Modelo Vista Controlador (MVC), entre los más populares se encuentran: Symfony, Ruby on Rails, CodeIgniter, Django, CakePHP, Zend Framework y Yii. Los mismos están basados en lenguajes de programación para el desarrollo web como PHP, Ruby y Python.

1.3.2. Patrón Modelo Vista Controlador

El MVC es un patrón que separa presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí. El componente Modelo maneja los datos del sistema y las operaciones asociadas a esos datos. El componente Vista define y gestiona como se presentan los datos al usuario. El componente Controlador dirige la interacción del usuario y pasa estas interacciones a Vista y Modelo (Sommerville, 2011).

Este patrón permite que los datos cambien de manera independiente de su representación y viceversa. Soporta en diferentes formas la presentación de los mismos datos, y los cambios en una representación se muestran en todos ellos.

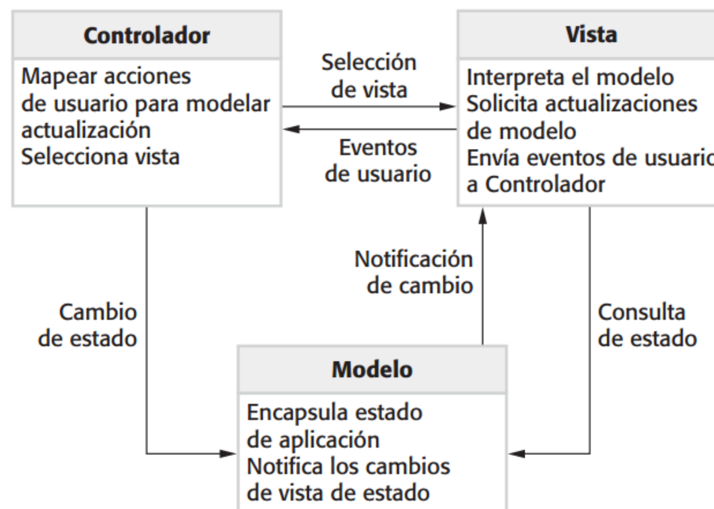

Figura 1: Patrón arquitectónico Modelo Vista Controlador.

El patrón MVC es el más utilizado en el desarrollo web, se muy flexible a cambios con respecto a opciones de multilenguaje y distintos diseños de presentación sin alterar la lógica de negocio. La separación de capas como presentación, lógica de negocio y acceso a datos es fundamental para el desarrollo de arquitecturas consistentes, reutilizables y fáciles de mantener, lo que resulta un ahorro de tiempo para el desarrollo de nuevos proyectos (Sommerville, 2011).

El uso del patrón MVC en aplicaciones web brindan facilidades para los desarrolladores, de tal manera ocurre con el uso de las metodologías de desarrollo de software como guía para el ciclo de vida de las mismas.

1.4. Metodologías de desarrollo de software

Para obtener un software con la calidad requerida, es necesario lograr un correcto ciclo de desarrollo, que permita organizar y efectuar satisfactoriamente los procesos que intervienen en la construcción del mismo. La correcta realización de estos procesos implica la utilización de metodologías de desarrollo que guíen la forma en que se aplica la Ingeniería de Software elevando la productividad del ciclo de desarrollo en aras de lograr un producto confiable y eficiente.

Rumbaugh plantea que *"una metodología de ingeniería de software es un proceso para la producción organizada del software, empleando para ello una colección de técnicas predefinidas y convencionales en las notaciones. Una metodología se presenta normalmente como una serie de pasos, con técnicas y notaciones asociadas a cada paso. Los pasos de la producción del software se organizan normalmente en un ciclo de vida consistente en varias fases de desarrollo"* (Rumbaugh, 2000).

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales o Pesadas (Noriega, 2015).

Existe una filosofía metodológica en el desarrollo de software denominado ágil, que proporciona mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque muestra su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales (Noriega, 2015).

Un paradigma de desarrollo de software es el desarrollo basado en modelos (MDD según sus siglas en inglés). El MDD subraya el uso de modelos en todos los niveles del software, este paradigma de desarrollo favorece los modelos sobre el código con el objetivo de reducir el tiempo de comercialización al tiempo que mejora la calidad del producto. El resultado de este cambio, trae consigo cambios significativos en la forma que el software es diseñado, mantenido y probado. Dentro de los MMD se encuentra el basado en prueba, la principal ventaja de estas técnicas de pruebas basadas en modelos es el incremento de la productividad y la calidad desplazando las actividades de prueba a una etapa anterior del proceso de desarrollo de software y la generación de casos de prueba que son independientes de cualquier aplicación particular del diseño. Existen muchas herramientas para cuantiosos enfoques de prueba basados en modelos, que varían

significativamente en sus diseños específicos pero ninguno de estos utiliza un documento de especificación de requisitos como artefacto de entrada (Mohamed Mussa, 2016).

Otro paradigma de desarrollo de software es la integración continua, que define un modo diferente de desarrollo. Requiere de una serie de buenas prácticas y la aceptación de las mismas por parte del equipo de desarrollo. Este enfoque está encaminado a los sistemas de producción e independientemente de la utilización de una metodología tradicional o ágil, las funcionalidades se van entregando poco a poco, y al final de cada iteración estas funcionalidades son integradas al sistema de producción real. Este enfoque se ajusta más a proyectos grandes o a líneas de producción donde se hacen entregas en el tiempo y es necesario probar extensamente un componente antes de integrarlo (Mohamed Mussa, 2016).

Actualmente, con el auge de la tecnología y con el objetivo de agilizar los procesos en el desarrollo de software, existe la necesidad de implantar metodologías de desarrollo de software que ayuden a entregar un producto de calidad en poco tiempo y costo estimado, las metodologías ágiles de desarrollo de software han despertado interés gracias a que proponen simplicidad y velocidad para crear sistemas. Las metodologías tradicionales no se adaptan a las nuevas necesidades o expectativas que tienen los usuarios hoy en día, debido que los métodos usados no son flexibles ante la posibilidad del surgimiento de nuevos requerimientos. Estos cambios generalmente implican altos costos, demanda de tiempo y la reestructuración total del proyecto que se esté llevando; en contraparte, los métodos ágiles permiten un desarrollo iterativo y adaptable que permite la integración de nuevas funcionalidades a lo largo del desarrollo del proyecto, para tanto el cliente como el equipo de desarrollo queden satisfechos.

1.5. Conclusiones parciales

El presente capítulo abordó lo referente al desarrollo de software, estableciendo la base teórica de la investigación y se realizó un estudio de las tendencias actuales, marcos de trabajo y metodologías de desarrollo de software. Después de tener estos elementos se arriba a las siguientes conclusiones:

Las tendencias analizadas en el desarrollo de software proporcionan un enfoque para abordar la solución al problema de la investigación.

El análisis de los marcos de trabajo ayudó a comprender las facilidades y patrones de diseños que estos brindan para resolver problemas enmarcados en el desarrollo de software y agilizar este proceso.

El análisis de las metodologías de desarrollo ofreció un enfoque al uso de metodologías ágiles para un proyecto de corta duración.

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DEL SISTEMA

2. Introducción

El presente capítulo hace uso de la metodología AUP-UCI para proponer la solución de la investigación. Se identifican los requisitos funcionales y no funcionales, así como la validación y análisis de los mismos. Por último, se realiza el diseño de la aplicación y la implementación de la misma.

2.1. Metodología de desarrollo de software

Según Somerville, para muchas personas el software es solo un programa de computadora. Sin embargo, comenta que son todos aquellos documentos asociados a la configuración de datos que se necesitan para hacer que estos programas operen de manera adecuada. Estos productos de software se desarrollan para algún cliente en particular o para un mercado en general. Para el diseño y desarrollo de proyectos de software se aplican metodologías, modelos y técnicas que permiten resolver el problema (Valdéz, 2014).

Metodología AUP-UCI

A partir del proceso de desarrollo de software llevado a cabo en la UCI enmarcado en el modelo CMMI que dicta las buenas prácticas que llevan a cabo este proceso, se selecciona una metodología para estandarizar las principales formas de hacer. Se basa en la metodología de Proceso Unificado Ágil (AUP), (según sus siglas en inglés), siguiendo los principios del programa de mejora basado en el modelo CMMI-DEV v1.3. Describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. La versión UCI, plantea 4 variantes para modelar negocios como, Caso de Uso del Negocio (CUN), Descripción de Proceso de Negocio (DPN), Modelo Conceptual (MC) e Historia de Usuario (HU). El equipo de trabajo utiliza la variante HU por la experiencia adquirida en proyectos similares, los cuales se necesitaba un desarrollo rápido, atendiendo que no es un negocio complejo, y además el cliente está presente durante el proceso de desarrollo (Sánchez, 2015).

Para darle solución al problema planteado en la presente investigación se hará uso de la metodología AUP-UCI, siguiendo sus tres fases exhibidas a continuación:

Inicio: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información

fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elabora la arquitectura y el diseño, se implementa y se libera el producto.

Cierre: en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

A continuación se realiza el desarrollo de las fases de la metodología:

2.2. Fase de Inicio

El objetivo de esta fase es obtener una comprensión común (cliente-equipo de desarrollo) del alcance del nuevo sistema y los preceptos fundamentales para entender el producto que se va a desarrollar. De este modo se prepara un marco conceptual para enmarcar las definiciones asociadas al negocio que se va a tratar para obtener un profundo entendimiento con el cliente y manejar los términos precisos con el objetivo de extraer la especificación de requisitos adecuada para darle solución al problema planteado en la investigación.

2.2.1. Calidad de software

La calidad de software es el grado en cual un proyecto puede definir hasta qué punto un software va a satisfacer las necesidades del usuario y del cliente y a la vez cuenta con la calidad requerida o definida por la entidad que desarrolla dicho software. A lo largo de los años varios autores han definido la calidad de software de las siguientes maneras.

Según la definición dada por la ISO-IEC 9126 la calidad de software es *"la totalidad de rasgos y atributos de un producto de software que le apoyan en su capacidad de satisfacer las necesidades del usuario dígase necesidades explícitas o implícitas"* (NC-ISO/IEC 9126, 1998).

Roger S. Pressman la define como *"la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente"* (Pressman, 2005).

Lo más interesante en estas dos definiciones de la Calidad de Software, es la necesidad de que un software de calidad debe satisfacer los requerimientos dados por el usuario. Reuniendo de forma conjunta las relaciones de conceptos expuestas anteriormente y para formar un criterio que sirva como fundamento para el desarrollo de esta investigación, se plantea de forma concreta que la calidad de software estaría dada por el conjunto de requisitos que necesariamente deben estar presente en un software de forma documentada con la capacidad de satisfacer las necesidades y requisitos del cliente obteniendo el mayor grado de satisfacción del mismo.

2.2.2. Pruebas funcionales de software

Una de las tareas más importantes en el desarrollo de productos de software son las pruebas, y cuando son aplicadas linealmente al ciclo de vida del producto desempeñan un papel crucial en el aseguramiento de la calidad. Buscando un mayor entendimiento del término Pruebas de Software se analizaron los siguientes conceptos:

Pressman afirma que las pruebas de software son la ejecución de una aplicación, o un trozo de código, para identificar uno o varios errores. Atendiendo a que el enfoque u objetivo de esta investigación es hacia las pruebas funcionales, se analizará solo como la ejecución de una aplicación (Pressman, 2005) .

La norma (9126-2) plantea que las Pruebas de Software no son más que aquellas actividades en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, donde los resultados se observan, registran y se realiza una evaluación de algún aspecto.

Pressman brinda una definición bastante menos directa que la dada por la norma (9126-2), enfocada a la calidad del software, la búsqueda de fallos y/o errores en general, Pressman además hace alusión que la prueba de software es un elemento crítico para la garantía de la calidad del software, las especificaciones, el diseño y la codificación, mientras que la (9126-2) es más específica, más enfocada al sistema de por sí.

Luego de analizar ambas definiciones se decide que el concepto más aceptado a utilizar para referirse a las pruebas funcionales de software sería: aquellas actividades donde un componente de un sistema es ejecutado con el fin de encontrar uno o varios errores para garantizar la calidad del software, sus especificaciones y diseño. Una de las maneras de ejecutar las pruebas funcionales es a través del método de prueba de caja negra, que son aquellas aplicadas al software y sus componentes.

2.2.3. Diseño de casos de prueba

El diseño de casos de prueba consiste en diseñar un conjunto de pruebas que permitan con mayor probabilidad detectar errores en el software. Las pruebas de caja negra, pretenden demostrar que las funciones del software son operativas, por tanto, se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, que la entrada se acepta de forma adecuada, que se produce una salida esperada y que la integridad de la información externa se mantiene. Esta examina algunos aspectos del modelo de prueba del sistema sin tener en cuenta la estructura interna del software. Se obtienen conjuntos de condiciones de entrada que ejerciten todos los requisitos funcionales del mismo. Las entradas son valores introducidos a la aplicación bajo prueba que pretenden la ejecución de una acción mientras que las salidas son los resultados esperados por la aplicación en respuesta a la acción ejecutada. Los escenarios, por su parte, son las combinaciones de valores de entradas que otorgarán determinadas respuestas. Los escenarios de pruebas comúnmente evaluados son el correcto, incorrecto e incompleto, en los que se basará la propuesta de solución presentada en esta investigación, esta se centrará en las pruebas de caja negra usando técnicas funcionales para aumentar la probabilidad de encontrar errores.

2.2.4. Técnicas de pruebas funcionales

Una vez que se ha implementado una aplicación, los desarrolladores necesitan verificar que está libre de anomalías y que se ha logrado el objetivo para la cual fue creada. Las pruebas de funcionalidad determinan la medida en la que la aplicación satisface los requisitos funcionales pautados. Los casos de prueba diseñados aplicando técnicas de caja negra pretenden encontrar errores asociados a:

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

De las técnicas existentes la presente investigación se centra en las siguientes: Partición de equivalencia y Tabla de Decisión en aras de satisfacer distintos objetivos de prueba.

2.2.5. Partición de equivalencia

Pressman define la partición equivalente dentro del método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requeriría la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar (Pressman, 2005).

Esta técnica define que cada caso debe cubrir el máximo número de entradas y debe tratarse el dominio de valores de entrada dividido en un número finito de clases de equivalencia que cumplan la siguiente propiedad: la prueba de un valor representativo de una clase permite suponer “razonablemente” que el resultado esperado (existan defectos o no) será el mismo que el obtenido, probando cualquier otro valor de la clase.

Mientras más entradas se tengan, más interesante se vuelve esta técnica, para los cuales se contemplan diversos datos de pruebas, relacionados entre sí, pues ellos generarán un mayor número de combinaciones para probar.

2.2.6. Tabla de decisión

La tabla de decisión es una técnica funcional que sintetiza procesos en los cuales se dan un conjunto de condiciones y de acciones a tomar según el valor de las condiciones. Puede utilizarse como herramienta en los distintos momentos del proyecto, esto es: en la exposición de los hechos, en el análisis del sistema actual, en el diseño del nuevo sistema y en el desarrollo del software (TABLADEC 2009, 2009).

La misma está integrada por: matriz de condiciones, matriz de acciones y matriz de reglas para condiciones y acciones. En la matriz de condiciones se enumeran todas las situaciones que pueden presentarse. Las reglas de condición indican qué valor debe asociarse a cada una de las condiciones. En la matriz de acciones se lista el conjunto de pasos que se debe seguir cuando se presentan ciertas condiciones. Las

reglas de acciones muestran las acciones específicas del conjunto que deben emprenderse dado los valores que toman las condiciones (TABLADEC 2009, 2009).

Esta técnica es fácil de comprender, arroja las posibles combinaciones a probar para un caso de prueba, útil para ganar tiempo en cualquier etapa del proyecto, agilizando el proceso de (Ibrahim, 2007) desarrollo de software.

2.2.7. Generación de casos de pruebas funcionales

Las pruebas funcionales son actividades donde un componente de un sistema es ejecutado con el fin de encontrar uno o varios errores, que permiten detectar en qué punto el producto no cumple con sus especificaciones y así comprobar su funcionalidad. Para realizarlas se debe hacer una planificación que consiste en definir los aspectos a examinar y la forma de verificar su correcto funcionamiento, punto en el cual adquieren sentido los casos de prueba (Palacio, 2009). Algunas metodologías han planteado varios métodos para la generación de casos de pruebas funcionales:

Heumann desarrolla un método para generar casos de prueba tomando como base casos de uso, e identificando dentro de cada uno los posibles escenarios o caminos de ejecución, y por último definir los valores a probar de cada caso de prueba. Finalmente se obtiene una lista de casos de prueba, con los valores que deben probar y los resultados esperados para cada caso (Heumann, 2001).

La propuesta de Riebisch está centrada en la transformación automática de un modelo de casos de uso a un modelo de uso que sirve como entrada para realizar pruebas estadísticas automáticas, que mejoran el nivel de cobertura, partiendo de que partes de un software no necesitan ser probadas con la misma minuciosidad. El método comienza con el refinamiento de los casos de uso ampliándolos con precondiciones y post-condiciones, alternativas al camino de ejecución principal y referencia a otros casos de uso relacionados. Después se traducen a diagramas de estado y se elabora el modelo de uso donde se indica la probabilidad de que ocurra una transición y se identifican los caminos de ejecución más frecuentes. Por último, se extraen los modelos de prueba a partir de los modelos de uso y se generan recorridos aleatorios sobre cada modelo de uso. Cada camino aleatorio será un caso de prueba (Matthias Riebisch, 2003).

La propuesta de (Ibrahim, 2007) presenta una herramienta llamada (GenTCase) para la generación de casos de pruebas. La herramienta permite dibujar un diagrama UML de requisitos funcionales y completar cada requisito funcional con un flujo de eventos y un diagrama de secuencia UML para cada requisito

funcional. El resultado es un archivo de texto con casos de pruebas generados. Sin embargo no se ha identificado que el diagrama de secuencia participe en el proceso de generación de pruebas.

De esta forma lo más ajustado para la presente investigación queda definido para el proceso de generación de casos de pruebas según (Palacio, 2009):

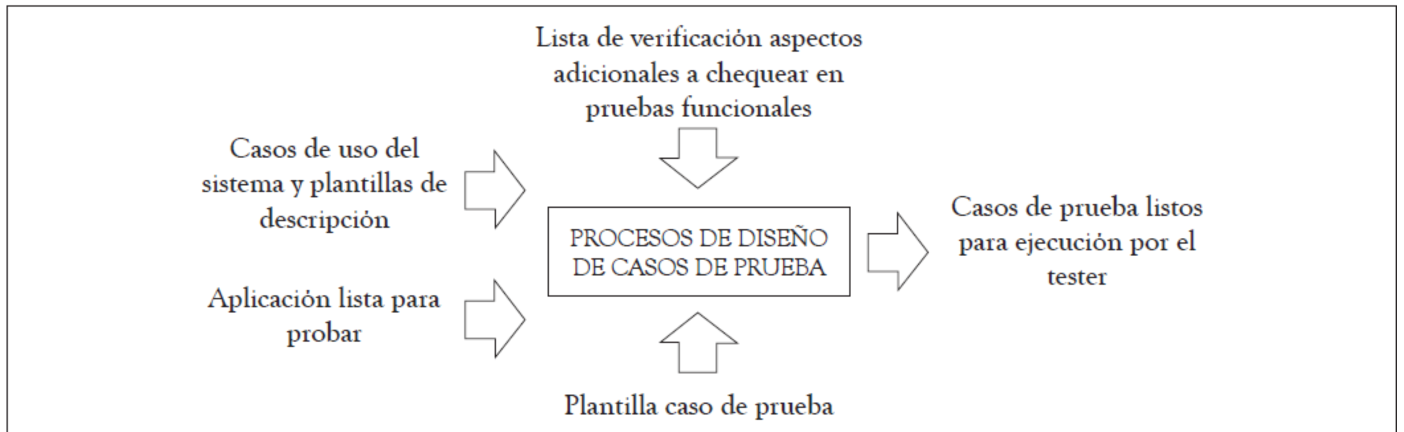

Figura 2: Proceso para la generación de casos de pruebas funcionales (Palacio, 2009).

2.3. Fase de Ejecución

El objetivo de la fase de ejecución es que el equipo de desarrollo profundice en la extracción de los requisitos del sistema y una vez obtenidos los mismos comenzar con el desarrollo de la aplicación.

2.3.1. Historia de usuario

Las historias de usuarios (HU) son la unidad mínima de agrupamiento equiparable a los casos de uso en RUP, con la diferencia de que deben ser escritos por el cliente usando un lenguaje sencillo. En esencia, son las ideas del cliente organizadas y agrupadas de acuerdo con su funcionalidad, estableciéndose un orden que permita priorizar sus necesidades, así como definir las que resultan críticas o claves en el momento de desarrollo de la solución. (Sánchez, 2015)

Durante la fase de ejecución se identificaron 17 HU representando cada una las funcionalidades del sistema, a continuación se muestran algunas debido a su importancia en el desarrollo de la aplicación:

Tabla 1: HU 1: Crear Caso de Prueba (Partición de Equivalencia).

Historia de Usuario	
Número: 1	**Nombre:** Crear Caso de Prueba (Partición de Equivalencia)
Iteración asignada: 1	
Prioridad en negocio: Alta (Alta / Media / Baja)	Puntos estimados: 1 semana
Riesgo en desarrollo: Alto (Alto / Medio / Bajo)	Puntos reales: 1 semana
Descripción: El sistema debe brindar la posibilidad al analista de generar casos de prueba a partir de la existencia de un documento de especificación de requisitos dentro de la aplicación.	
Observaciones: El analista, una vez que esté adicionando un documento de especificación de requisitos, generará los casos de pruebas con la salva del documento de especificación de requisitos.	

Tabla 2: HU 2: Crear Caso de Prueba (Tabla de decisión).

Historia de Usuario	
Número: 15	**Nombre:** Crear Caso de Prueba (Tabla de decisión)
Iteración asignada: 1	
Prioridad en negocio: Alta (Alta / Media / Baja)	Puntos estimados: 2 semanas
Riesgo en desarrollo: Alto	Puntos reales: 2 semanas

(Alto / Medio / Bajo)	
Descripción: El sistema debe brindar la posibilidad al usuario de genere casos de prueba utilizando la técnica Tabla de decisión.	
Observaciones: El usuario, una vez que esté insertado el Documento ERS, señalará dentro de uno de los módulos qué requisito será al cual se aplique la técnica de prueba Tabla de decisión.	

2.3.2. Requisitos funcionales y no funcionales

Los requisitos son la base para un desarrollo exitoso, así como para una plena conformidad con el entregable final. A continuación, se muestra el listado de requisitos definidos para el sistema informático propuesto por la investigación.

Gestionar Usuario:

RF1: Crear Usuario.

RF2: Editar Usuario.

RF3: Eliminar Usuario.

RF4: Listar Usuario.

Gestionar Proyecto:

RF5: Crear Proyecto.

RF6: Editar Proyecto.

RF7: Listar Proyecto.

Confeccionar el documento de especificación de requisitos de software:

RF8: Confeccionar Presentación.

RF9: Confeccionar Control del Documento.

RF10: Confeccionar Introducción.

RF11: Confeccionar Catálogo de Módulos.

RF12: Confeccionar Requisitos.

RF 13: Exportar Documento a PDF.

Generar Casos de Pruebas:

RF14: Agregar documento de especificación de requisito.

RF15: Modificar documento de especificación de requisito.

Técnica Partición de Equivalencia:

RF16: Generar Caso de Prueba para la Técnica Partición de Equivalencia.

Técnica Tabla de Decisión:

RF17: Generar Caso de Prueba para la Técnica Tabla de Decisión.

Requisitos no funcionales:

Los requisitos no funcionales (RNF) definen propiedades del sistema informático que se desea como producto final. Estos requisitos son normalmente a los que debe apuntar la arquitectura, de no cumplirse, el sistema informático propuesto puede no funcionar o el cliente simplemente no aceptará el producto.

Seguridad:

RNF1: Administración de permisos. Los usuarios registrados en el sistema podrán acceder a funcionalidades específicas en dependencia de los permisos pertenecientes a su rol.

Usabilidad:

RNF2: Diseño. Los iconos usados son representativos del lenguaje del dominio de la aplicación. Los nombres de etiquetas y las informaciones están asociados también a este dominio.

RNF3: Arquitectura de información. Para realizar cualquier acción de gestión del sistema no se debe dar más de 3 clic, y las funcionalidades se agrupan en módulos atendiendo a la relación existente entre ellas.

Soporte

RNF4: Para garantizar un sistema mantenible, reparable y escalable se desarrollará orientado a componentes, es decir por bloques de construcción que conformarán las partes del sistema. Dichos componentes están representados por módulos y capas de abstracción que garanticen un código cohesionado con las responsabilidades bien delimitadas. Se implementará la infraestructura base de la arquitectura que siga las pautas de diseño y codificación para garantizar la integración de los componentes como un todo.

Requisitos de hardware

RNF5: Las condiciones mínimas requeridas para el servidor son: un procesador Dual Core 1.2 Ghz con 1 Gb de memoria RAM, un disco duro de 80 Gb y conexión de red a 100 Mb/s.

RNF9: Las condiciones mínimas requeridas para el cliente son: un procesador Dual Core 1.0 Ghz con 512 Mb de memoria RAM, un disco duro de 40 Gb y conexión de red a 100 Mb/s.

Requisitos de software

RNF6: Las condiciones mínimas requeridas son: un explorador web, ya sea, Mozilla Firefox v36, Google Chrome v39 o Internet Explorer 9.

2.3.2. Validación de requisitos

Lista de Chequeo:

La validación es la comprobación de un conjunto de datos para determinar si su valor se halla dentro de unos límites de fiabilidad. Es una actividad sustancial para el proceso de desarrollo de software, logrando una disminución de errores y costos de mantenimiento.

La lista de chequeo es una herramienta que funciona como una lista de factores claves para validar los requisitos de software. Acentuar que la lista de chequeo debe estar libre de errores, por tanto, garantiza que todos los requerimientos sigan los estándares de calidad que cuya lista define. A continuación, se expresan dichos estándares:

Completo: Un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.

No ambiguo: Un requerimiento no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.

Probado: Un requisito es probado si y solo si, existe un proceso finito por el cual una máquina o persona puede chequear que el producto de software completa el requisito.

Modificable: Cualquier cambio puede realizarse de manera sencilla, completamente y consistentemente manteniendo estructura y estilo.

Único: El requisito debe tener un identificador único.

Consistente: El requisito es congruente con otros requisitos relacionados.

Correcto: Cada requisito escrito es uno que el software tiene que cumplir.

Traceable: El origen de cada requisito está claro y la Especificación de Requisitos de Software facilita la referencia a cada requisito en desarrollos futuros o mejoras de documentos.

Con esta herramienta fueron validados los requisitos propuestos por el cliente, según los parámetros anteriormente descritos y el 100% de están listos para ser implementados.

2.4. Descripción de la propuesta de solución

El sistema propuesto pretende desarrollar una herramienta para generar Casos de Pruebas, a partir de las técnicas funcionales Partición de equivalencia y Tabla de decisión, mediante el documento de especificación de requisitos de los proyectos, facilitando el trabajo de los analistas en los proyectos de CEGEL y mejorar la elaboración de este artefacto. Una vez introducido el documento de especificación de requisitos en la aplicación, se muestran los módulos y requisitos referentes al mismo, de tal forma que cada requisito tiene asociado un caso de prueba a mostrar con la técnica de prueba funcional Partición de Equivalencia o Tabla de Decisión generado una vez que el mismo esté dentro del sistema.

2.4.1. Prototipos

A continuación, se muestran los prototipos de interfaz, estos responden a las historias de usuarios anteriormente mostradas, las cuales están asociadas a los requisitos Generar Caso de Prueba (Partición de Equivalencia) y Generar Caso de Prueba (Tabla de Decisión), que son los de mayor prioridad para el desarrollo:

Generar Caso de Prueba (Partición de Equivalencia):

Caso de Prueba | Descripción de Variables

Título:

Escenarios	Descripción	Var1	Var2	Var3	Respuesta del Sistema	Flujo Central
Correcto						
Incorrecto						
Incompleto						

Cerrar   Guardar

Figura 3: Crear Caso de Prueba con Partición de Equivalencia.

Generar Caso de Prueba (Tabla de decisión):

Caso de Prueba

PreCondiciones	CP1	CP2	CP3	CP4
Precondición 1				
Precondición 2				
Precondición 3				

PosCondiciones	CP1	CP2	CP3	CP4
Poscondición 1				
Poscondición 2				
Poscondición 3				

Cerrar Guardar

Figura 4: Crear Caso de Prueba con Tabla de Decisión.

Estos prototipos sirven para guiar al desarrollador en la implementación de la herramienta, ilustrando las interfaces de los requisitos de mayor peso para la investigación.

2.5. Selección de tecnologías

Visual Paradigm

Herramienta CASE de modelado UML que proporciona 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. Es capaz de soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de diferentes tipos de diagramas. Por tanto es utilizada para generar los artefactos necesarios para el desarrollo de la investigación (VP-UML, 2016).

Esta herramienta está disponible en múltiples plataformas (Windows, GNU/Linux), su diseño se centra en casos de usos y se enfoca a la calidad del software. Es fácil de instalar y actualizar, es distribuida bajo licencia libre, tiene capacidad de ingeniería directa e inversa, soporta UML y ORM, y sus versiones son compatibles entre sí. Brinda funcionalidades para la generación de código fuente a partir de diagramas de clases (VP-UML, 2016).

PHP

El lenguaje PHP (cuyo nombre es acrónimo de Hypertext Preprocessor) es un lenguaje interpretado con una sintaxis similar a la de C++ o JAVA. Aunque el lenguaje se puede utilizar para realizar cualquier tipo de programa, es en la generación dinámica de páginas web donde ha alcanzado su máxima popularidad. En concreto, suele incluirse embebido en páginas HTML o XHTML, siendo el servidor web el encargado de ejecutarlo. El estudio comparativo del lenguaje de programación estuvo condicionado por el conocimiento del equipo de desarrollo, teniendo en cuenta la premura del cliente referente al uso del sistema. Es por esto que, a pesar de existir diversos lenguajes, con características importantes, se escoge PHP, con el objetivo de ganar en tiempo, aprovechar la facilidad de aprendizaje, las novedades que aporta y la necesidad de crear una aplicación web (PHP, 2016).

PHPStorm

JetBrains PhpStorm es un IDE multi-plataforma para PHP desarrollado sobre la plataforma JetBrains IntelliJ IDEA. Proporciona un editor para PHP, HTML y JavaScript con el análisis de código en la marcha con el objetivo de prevenir errores y refactorizaciones automáticas para PHP y JavaScript. Soporta PHP 5.3, 5.4, 5.5 y 5.6 para crear proyectos modernos y heredados incluyendo un editor de SQL con resultados de la consulta editable. Los usuarios pueden ampliar el IDE mediante la instalación de plugins creados para la plataforma de IntelliJ o escribir sus propios plugins. PhpStorm trae incluido un driver para trabajar directamente con la aplicación PHPMyAdmin, por tal motivo la misma se escoge para la gestión de base de datos (jetbrains, 2016).

PHPMyAdmin

Se trata de una aplicación de software libre que ofrece un entorno amigable a la hora de trabajar con las bases de datos. Permite trabajar bien con SQL (el lenguaje propio de muchas bases de datos, entre ellas MySQL), o bien con otro más sencillo e intuitivo (MyAdmin, 2016).

Symfony2.7

Symfony es un marco de trabajo*,* es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que pueden servir de base para la organización y desarrollo de un software.

Symfony es un framework construido sobre la base del lenguaje PHP con varios componentes independientes que facilita la creación de sitios y aplicaciones web. Utilizar Symfony es gratuito, incluso aunque se creen aplicaciones y programas que después se vendan a otros clientes. A pesar de esta

gratuidad, se ha desarrollado un ecosistema de empresas que viven de Symfony gracias a la formación, consultoría y desarrollo de proyectos. Su código, y el de todos los componentes y librerías que incluye, se publican bajo la licencia MIT de software libre. La documentación del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos, permitiendo crear aplicaciones desde las más simples hasta lo más complejo que se quiera. Los componentes de Symfony son tan útiles y están tan probados, que proyectos tan gigantescos como Drupal 8 están construidos con ellos (Symfony, 2016).

Twig

Twig es un motor para crear plantillas con lenguaje PHP, es un amigable ambiente para el diseñador y desarrollador apegado a los principios de PHP, añadiendo útiles funcionalidades a los entornos de plantillas (Sensiolabs, 2016).

Las características clave son:

- **Rápido:** Twig compila las plantillas hasta código PHP regular optimizado. El costo general en comparación con código PHP regular se ha reducido al mínimo.
- **Seguro:** Twig tiene un modo de recinto de seguridad para evaluar el código de plantilla que no es confiable. Esto permite utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.
- **Flexible:** Twig es alimentado por flexibles analizadores léxico y sintáctico. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizados, y crear su propio DSL.

Twig ofrece la ventaja de separar completamente la lógica de control con la lógica de vistas, donde al diseñador no le va a llegar ninguna información del controlador y podrá ser libre de diseñar sin necesidad de tener algún conocimiento de la aplicación.

TortoiseGit

TortoiseGit es un cliente de código abierto para el sistema de control de versiones Git, es decir, gestiona los archivos con el tiempo y los almacena en un depósito local. El repositorio es como un servidor de archivos ordinario, excepto que recuerda todos los cambios hechos a sus archivos y directorios. Esto le permite recuperar versiones antiguas de sus archivos y examinar la historia de cómo y cuándo cambiaron sus datos, y quién lo cambió. Por esta razón, Git y los sistemas de control de versiones en general son vistos como una especie de "máquina del tiempo". TortoiseGit es utilizado con el propósito de mantener un control de versiones con respecto al código generados por los programadores en la presente investigación (Tortoisegit, 2016).

2.6. Diagrama de clases del diseño

Los diagramas de clases representan las relaciones existentes entre las clases de un sistema. Se emplean para modelar la vista estática del diseño de un sistema, visualizar, especificar y documentar modelos estructurales, para construir sistemas ejecutables.

La estructura que guiará el desarrollo de la aplicación tendrá como base el siguiente diagrama de clases:

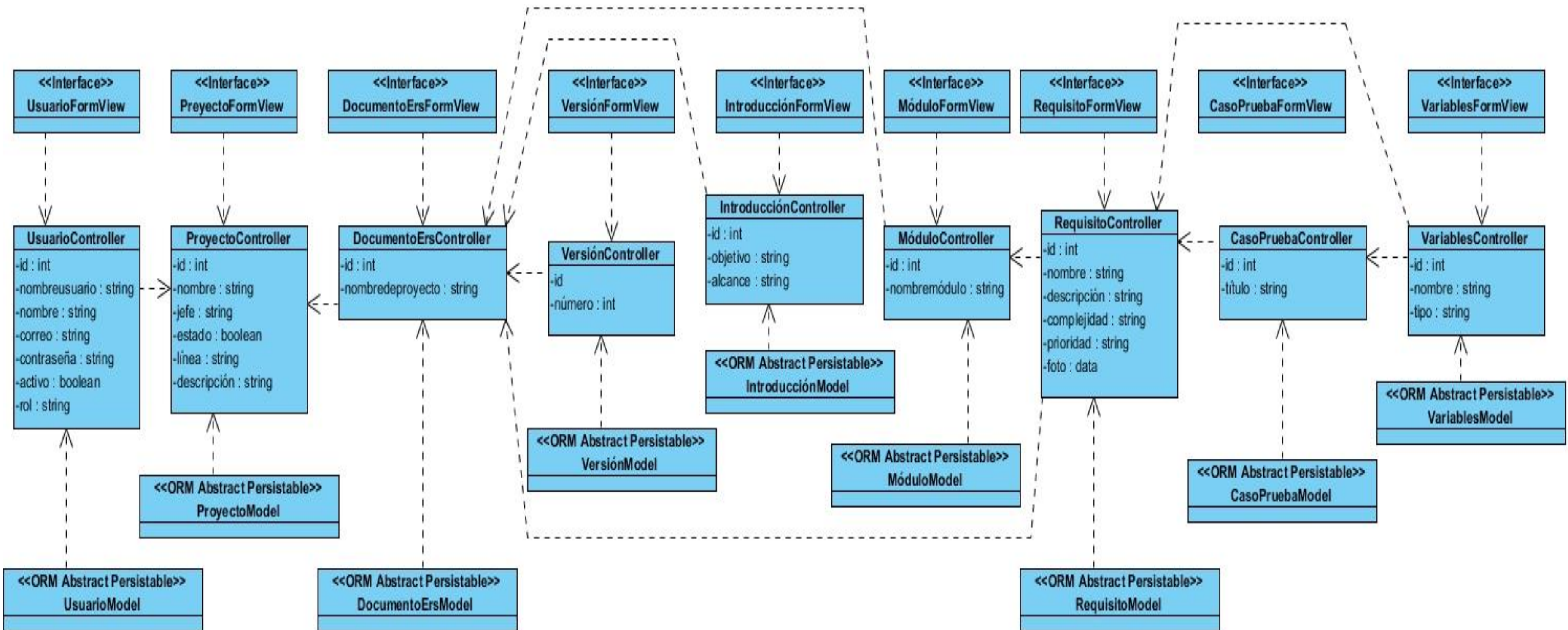

Figura 5: Diagrama de Clases del Diseño.

2.6.1. Diseño arquitectónico

La metodología de desarrollo de software AUP-UCI sugiere que se realicen diseños simples y sencillos, para de esta forma lograr que sean de fácil entendimiento en la fase de implementación, lo que le costará menos tiempo al desarrollador llevar a cabo esta tarea.

Descripción de la Arquitectura

Arquitectura n-capas

La arquitectura n-capas soporta el desarrollo incremental del sistema. Conforme se desarrolla una capa, alguno de los servicios proporcionados por esta, deben quedar a disposición de los usuarios. En tanto su interfaz no varía, una capa puede sustituirse por otra equivalente. Cuando una capa cambia o se agregan nuevas funcionalidades, solo resulta afectada la capa adyacente (Sommerville, 2011).

Esta arquitectura Organiza el sistema en capas con funcionalidades relacionadas con cada capa. Una capa ofrece servicios a la capa inmediatamente superior, de modo que las capas de nivel inferior representan servicios de núcleo. La solución propuesta será una versión inicial a la satisfacción de las necesidades de los clientes. Planteada esta situación, se decide emplear una arquitectura n-capas. Teniendo en cuenta las futuras necesidades de los usuarios, se podrán modificar cualquiera de sus componentes sin tener que afectar en gran medida el resto de los mismos, solo los que se encuentran en su nivel.

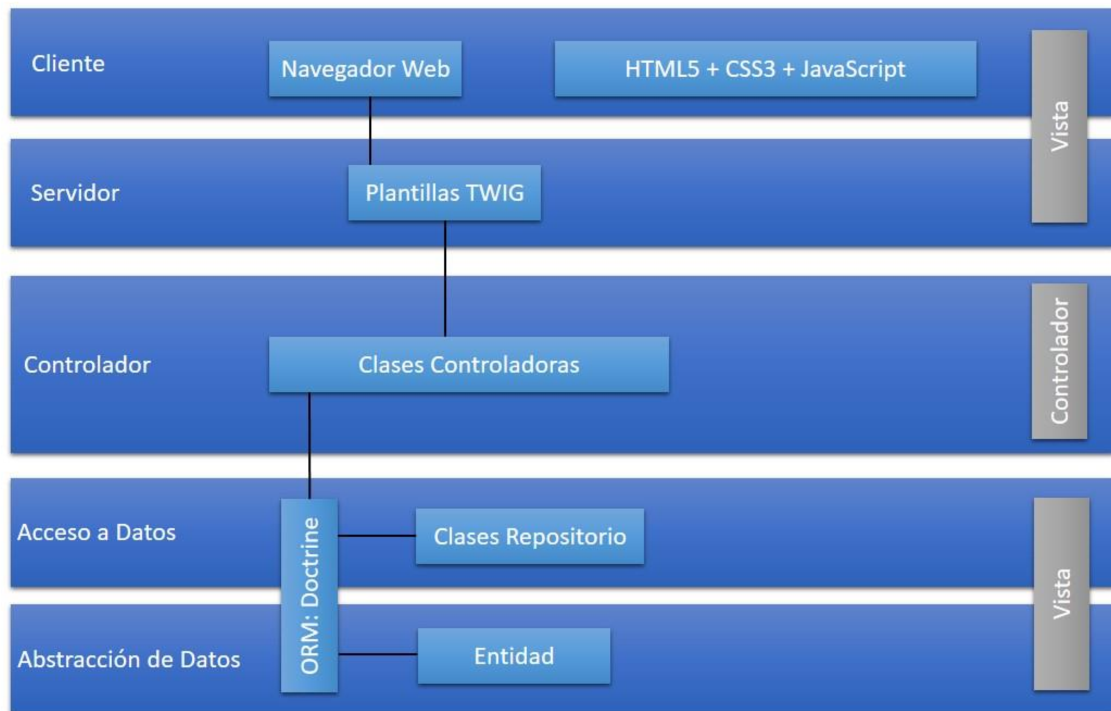

Figura 6: Esquema de la arquitectura del sistema.

Patrón MVC

Conjuntamente implementado en la estructura de Symfony se utiliza el patrón arquitectónico (MVC), separando la lógica del negocio, del modelo y la vista, por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo y la vista de los detalles del protocolo (HTTP) del inglés Hyper Text Transfer Protocol, utilizado para las peticiones. El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación. A continuación se muestra una breve descripción de cada una de las capas de la arquitectura definida.

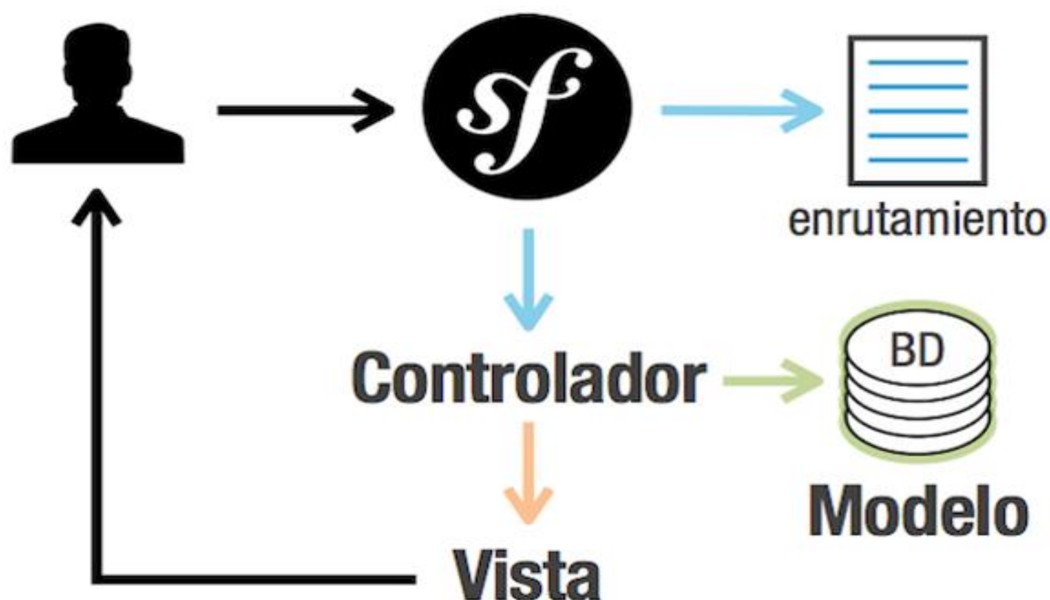

Figura 7: Diagrama MVC.

Vista

Esta capa presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso. Esta capa se comunica únicamente con la capa del controlador. También es conocida como interfaz gráfica y debe tener la característica de ser amigable (entendible y fácil de usar) para el usuario. Esta capa contiene los componentes que implementan y muestran la interfaz de usuario, según sus siglas en inglés (UI), además administra la interacción con el usuario y ejecuta la lógica asociada con la validación de los datos. Los componentes de UI proporcionan una forma para que los usuarios interactúen con la aplicación, mostrando los datos en el formato adecuado. Asimismo, obtienen y validan los datos entrados por el usuario.

Controlador

En esta capa se encuentran las clases controladoras o clases *Action* de la aplicación las cuales se encargan de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. En ella también se halla el Controlador Frontal, este es un componente que sólo tiene código relativo al MVC, por lo que no es necesario crear uno, ya que Symfony lo genera de forma automática.

Modelo

Esta capa se encuentra dividida en dos, la capa de Acceso a Datos la cual se encarga de la gestión de la información y la capa de Abstracción de Datos en la cual se hallan las clases entidades y el mapeo real de la base de datos. Dichas capas se generan automáticamente, en función de la estructura de datos de la aplicación. El Object Relationship Manager, según sus siglas en inglés (ORM) se encarga de crear la estructura básica de las clases y genera automáticamente todo el código necesario.

La abstracción de la base de datos es completamente transparente para el programador, se realiza de forma nativa mediante PDO (PHP Data Objects). Así, si se cambia el sistema gestor de bases de datos en cualquier momento, no se debe reescribir ni una línea de código, ya que tan sólo es necesario modificar un parámetro en un archivo de configuración.

Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a 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. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Patrones GRASP

Los patrones generales de software para asignar responsabilidades del inglés (General Responsibility Assignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Larman, 1999). Existe un gran número de patrones que se sitúan dentro de este grupo. A continuación se señalan.

Experto

Asigna una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Este es uno de los patrones que se implementa al utilizar las clases controladoras o clases *Action* de la aplicación para la gestión de formularios en el sistema.

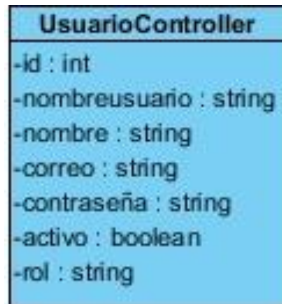

Figura 8: Patrón Experto.

Alta cohesión

Asigna una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme (Larman, 1999).

Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. No conviene este tipo de clases pues presentan los siguientes problemas:

- Las clases son complejas de comprender.
- Las clases son difíciles de reutilizar y conservar.

Las clases con baja cohesión a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos.

Una de las características principales del framework Symfony es la organización en cuanto a la estructura del proyecto, lo cual permite crear y trabajar con clases con una alta cohesión. Por ejemplo, la clase *Action* contiene varias funcionalidades estrechamente relacionadas entre ellas, teniendo un sentido común y un propósito único, siendo estas las encargadas de controlar las acciones de las plantillas. Esto hace posible que el software sea flexible a cambios sustanciales con efecto mínimo.

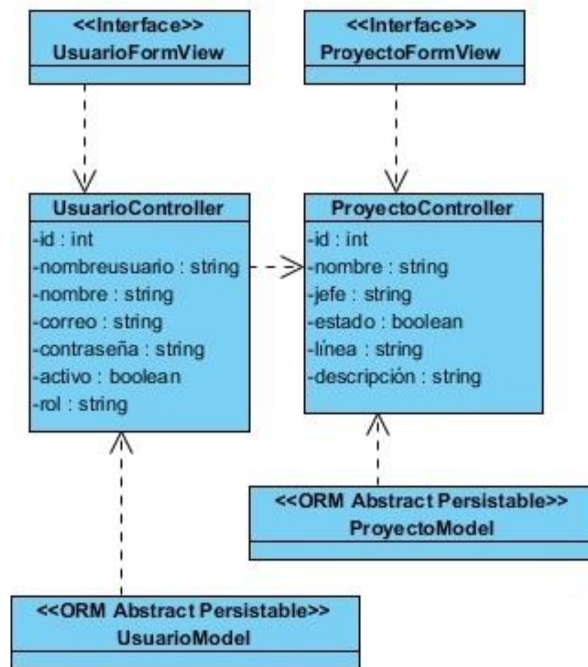

Figura 9: Patrón Alta Cohesión.

Bajo acoplamiento

Asigna una responsabilidad para mantener bajo acoplamiento. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, las conoce y/o recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras.

Este patrón está evidenciado en el framework Symfony ya que dentro de la capa modelo las clases de abstracción de datos son las más reutilizables y no tienen asociaciones con las clases de la capa vista.

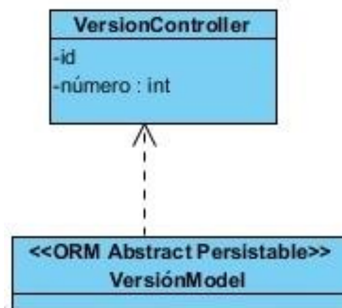

Figura 10: Patrón Bajo Acoplamiento.

Controlador

Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento. Un ejemplo de ellos son las clases controladoras empleadas para la implementación del sistema.

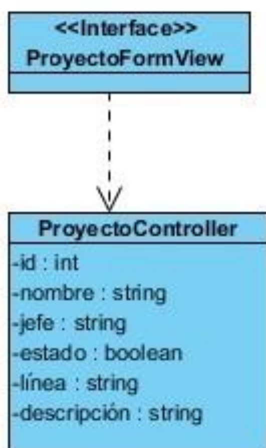

Figura 11: Patrón Controlador.

Modelo de Datos

La creación del modelo de datos constituye prioridad en la abstracción para la creación de la base de datos siendo posible esto a través de un grupo de herramientas las cuales describen los datos y las relaciones que existen entre ellos. Haciendo uso del diagrama entidad relación de la herramienta CASE Visual Paradigm para UML se realizó el siguiente modelo de datos:

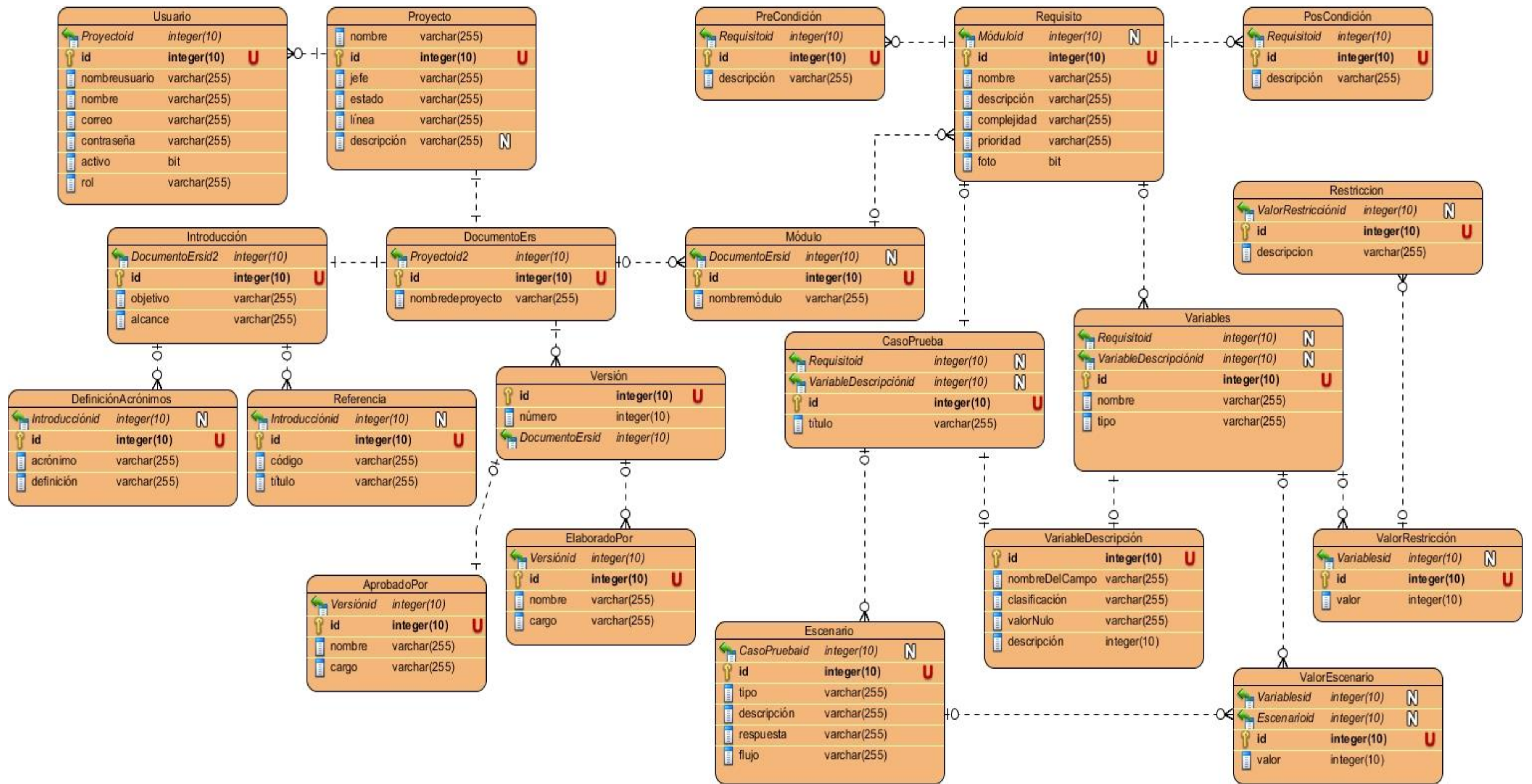

Figura 12: Modelo de Datos Relacional.

El modelo de datos del sistema cuenta con 20 tablas o entidades necesarias para la gestión de los documentos de especificación de requisitos. Dicho modelo describe las entidades DocumentoErs, Versión, Introducción, Módulos como las entidades necesarias para la creación de un documento de especificación de requisitos.

2.7. Implementación

La implementación del sistema es la parte más importante dentro del avance del proyecto en la fase de ejecución de la metodología AUP-UCI. Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información (Sánchez, 2015).

En una iteración se repite el flujo de trabajo de las disciplinas, Requisitos, Análisis y diseño, Implementación y Pruebas internas. De esta forma se brinda un resultado más completo para un producto final de manera creciente. Para lograr esto, cada requisito debe tener un completo desarrollo en una única iteración.

2.7.1. Estándares de codificación

Los estándares de codificación son aquellos que permiten entender de manera rápida y sencilla el código empleado en el desarrollo de un software. Garantizan el mantenimiento óptimo de dicho código por parte del programador (Calleja, 2009). A continuación se muestran algunas pautas del estándar definido por el equipo de desarrollo así como ejemplos de su uso:

- Para los nombres de los métodos, en caso de que estos se nombren con una sola palabra, esta se escribe en minúsculas y en caso de ser un nombre compuesto, las palabras que lo conforman se escriben juntas, de la segunda en adelante se escriben con letra inicial mayúscula. Se emplea la notación Camel variante (LowerCamelCase).

Ejemplo: *private function createCreateForm () {…}*

- Los nombres de las clases se escriben con la primera letra de cada palabra que lo compone en mayúscula, haciendo uso de la notación Camel, con la variante (UpperCamelCase).

Ejemplo: *class FosUserModel {…}*

- Las clases que se encuentran dentro de la carpeta **Controller** después del nombre de la clase llevan la palabra: “Action”.

Ejemplo: *public function showAction () {…}*

- Las clases formularios comienzan con el nombre del formulario según su función, seguido de la palabra “Type” y todas extienden de la clase “AbstractType”.

Ejemplo: *class ModuloType extends AbstractType {…}*

2.8. Conclusiones parciales

En este capítulo se abordó lo referente a la fase de Ejecución donde se definieron los requisitos funcionales, no funcionales y las HU, permitiendo así definir las propiedades y bases para el desarrollo del sistema a implementar.

Se aborda la arquitectura, los patrones de diseño, y el modelo relacional para brindar un mayor entendimiento, organización y claridad para la implementación de la solución, lo cual permite que futuras mejoras en la programación sean menos engorrosas.

CAPÍTULO 3: VERIFICACIÓN Y VALIDACIÓN

3. Introducción

Se utiliza en el capítulo la tercera fase de la metodología AUP-UCI para llevar a cabo el proceso de pruebas utilizando métricas para validar el funcionamiento de todos los requisitos identificados y el diseño planteado para el desarrollo del sistema informático. Para la verificación del sistema se realizan pruebas de caja negra examinando las funcionalidades a nivel de interfaces en el sistema y se realiza la validación de las variables que se plantearon en el problema a resolver.

3.1. Fase de cierre

El objetivo de esta fase es llevar el sistema a entornos donde se somete a pruebas de validación y aceptación, analizar los resultados y finalmente se despliega en los sistemas de producción. A continuación se da inicio a la fase de cierre de la metodología AUP-UCI.

3.2. Métricas

Las métricas del software son una medida cuantitativa de evaluar la calidad de los atributos internos de un sistema. Se emplean con el objetivo de llevar el control de la calidad del producto que se está desarrollando, evaluar la efectividad del proceso y mejorar la calidad del trabajo. Las métricas proporcionan los conocimientos necesarios para crear modelos efectivos de análisis y diseño, un código sólido y pruebas exhaustivas (Pressman, 2005).

3.2.1. Métricas para validar el diseño

Tamaño Operacional de Clase (TOC)

Al aplicar la métrica TOC se tuvieron en cuenta un conjunto de atributos de calidad que se relacionan a continuación:

- **Responsabilidad:** consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio.

- **Complejidad de implementación:** consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

- **Reutilización:** consiste en el grado de reutilización presente en una clase o estructura de clases, dentro de un diseño de software.

Para determinar el valor de los atributos de calidad, se debe determinar la cantidad de procedimientos (CP) que posee cada una de las clases a medir. Una vez determinado el CP se procede a calcular el promedio del mismo y se determina la incidencia de los atributos de calidad en cada una de las operaciones de las clases.

La aplicación del instrumento de evaluación de la métrica TOC para el número total de operaciones fue desarrollada para una muestra de 12 clases escogidas aleatoriamente. A continuación se muestran los resultados obtenidos:

Tabla 3: Relación de números de procedimiento por clase.

Clases	Cantidad de Procedimientos
AprobadoPorController	11
CasoPruebaController	9
DefinicionAcronimosController	7
DocumentoErs	15
ElaboradoPor	8
Escenario	15
FosUserType	3
IntroduccionType	3
ModuloExcludeRequisitoType	3
PosCondicionModel	6
PreCondicionModel	6
ProyectoModel	6

Resultado de la aplicación de la métrica TOC

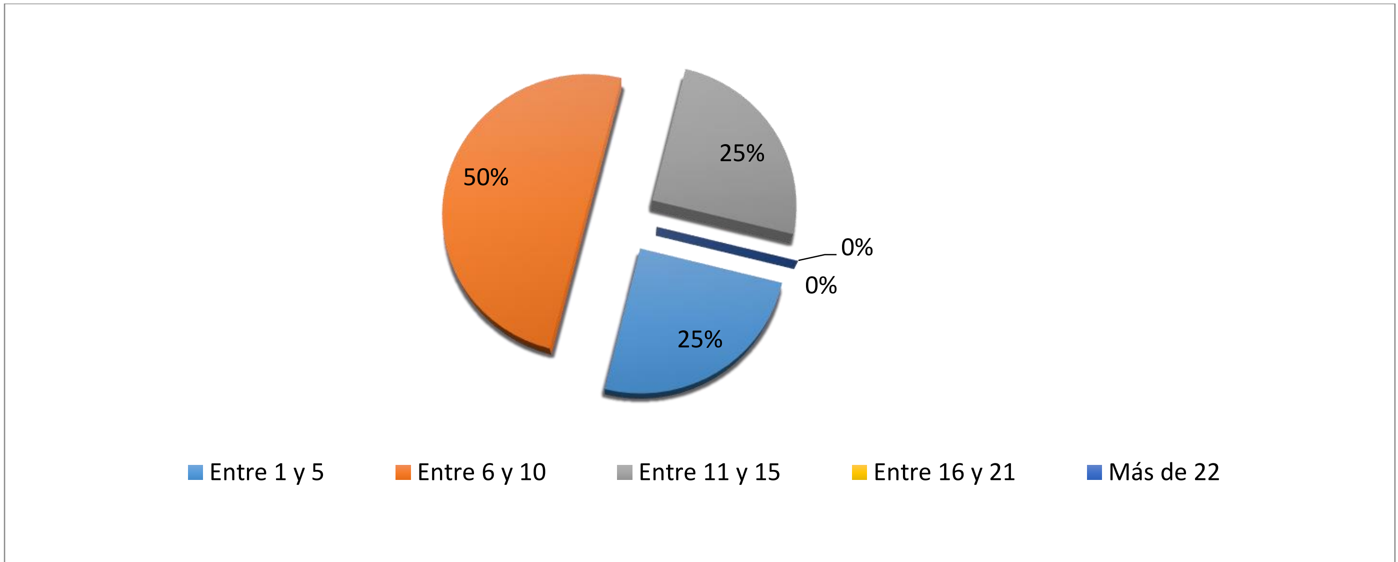

Figura 13: Representación en porciento de los resultados obtenidos, obtenidos tras aplicar la métrica TOC.

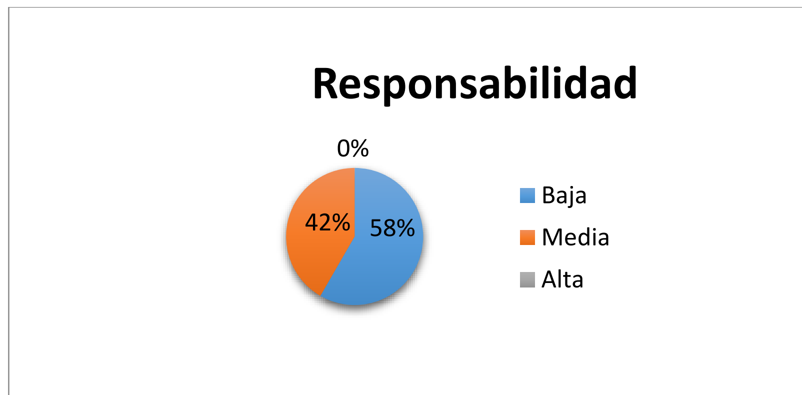

Figura 14: Valor del atributo de calidad Responsabilidad.

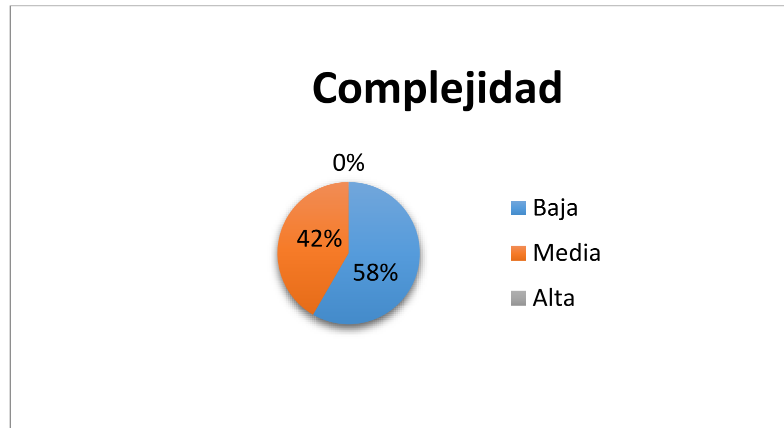

Figura 15: Valor del atributo de calidad Complejidad de implementación.

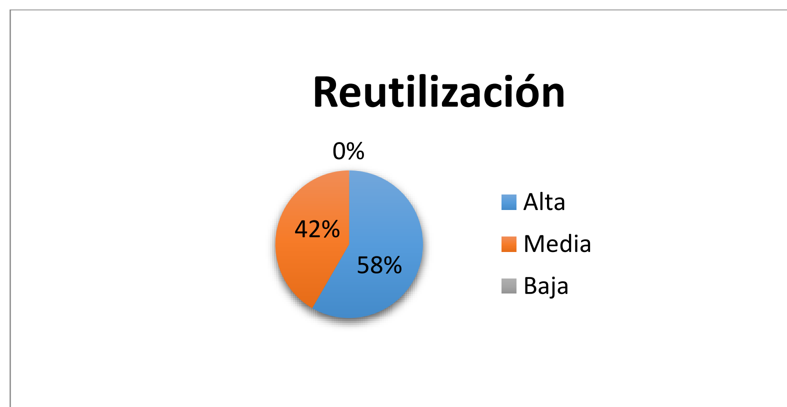

Figura 16: Valor del atributo de calidad Reutilización.

Responsabilidad: luego de aplicar la métrica se obtuvieron resultados satisfactorios que reflejan una responsabilidad baja con valor del 58%.

Complejidad de Implementación: después de haberse realizado la medición de la métrica, arrojó resultados positivos ya que la complejidad de las clases es baja en un 58%.

Reutilización: se obtuvieron valores en un nivel alto con un 58%.

Haciendo un análisis de los resultados obtenidos para los atributos de la métrica TOC se puede observar que el atributo reutilización cuenta con un porciento alto, demostrando así que el componente cuenta con una elevada reutilización, baja responsabilidad y complejidad en el diseño propuesto. Por lo que se concluye que los resultados obtenidos en esta métrica son positivos.

Relaciones entre Clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra. Permite evaluar el acoplamiento, la complejidad de mantenimiento, la reutilización y la cantidad de pruebas de unidad necesarias para probar una clase, teniendo en cuenta las relaciones existentes entre ellas.

- **Acoplamiento:** consiste en el grado de dependencia o interconexión de una clase o estructura de clases con otras, está muy ligada a la característica de reutilización.
- **Complejidad de mantenimiento:** consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas:** un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Para determinar el grado de afectación de los atributos de calidad que mide la métrica RC es necesario determinar la cantidad de relaciones de uso (CRU) que posee cada una de las clases a medir.

Una vez determinada la CRU, se procede a calcular el promedio de las mismas y teniendo ambos valores se determina la incidencia de los atributos de calidad en cada una de las clases.

A continuación, se muestran los resultados obtenidos de dicha métrica para una muestra de 12 clases:

Tabla 4: Relaciones de uso entre clases.

Clases		Cantidad de Relaciones de Uso
AprobadoPorController		7
CasoPruebaController		7
DefinicionAcronimosController		7
DocumentoErs		2
ElaboradoPor		1
Escenario		1

FosUserType		2
IntroduccionType		3
ModuloExcludeRequisitoType		3
PosCondicionModel		2
PreCondicionModel		2
ProyectoModel		2

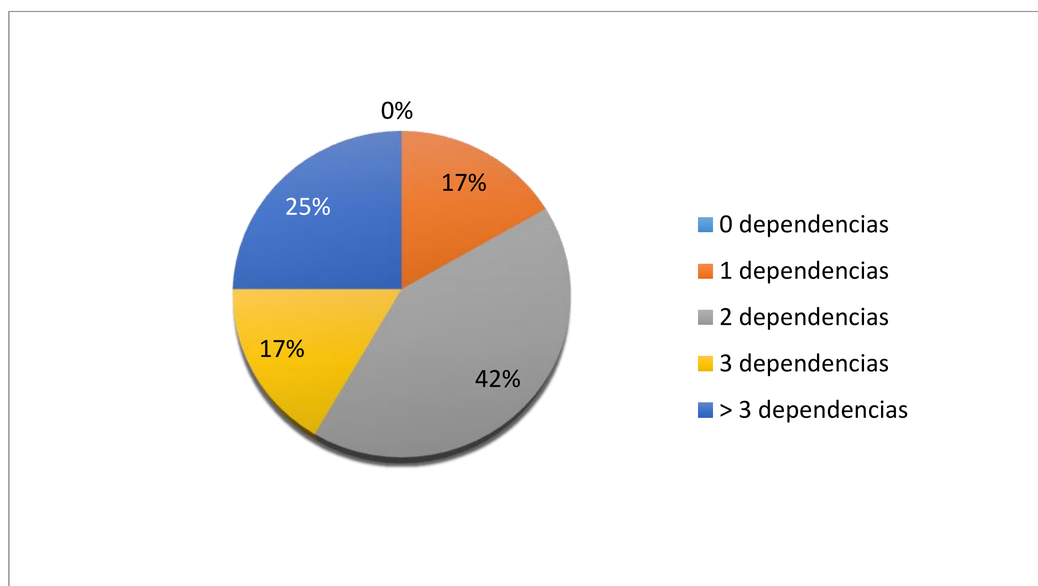

Figura 17: Representación en porciento de la aplicación de la métrica RC en intervalos definidos.

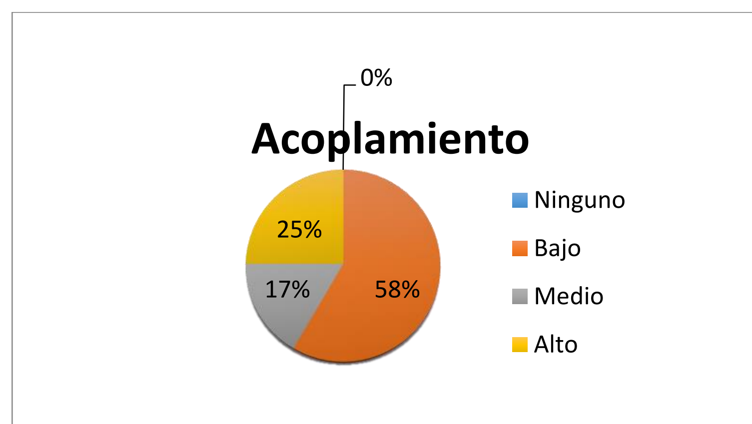

Figura 18: Grado de afectación del atributo Acoplamiento.

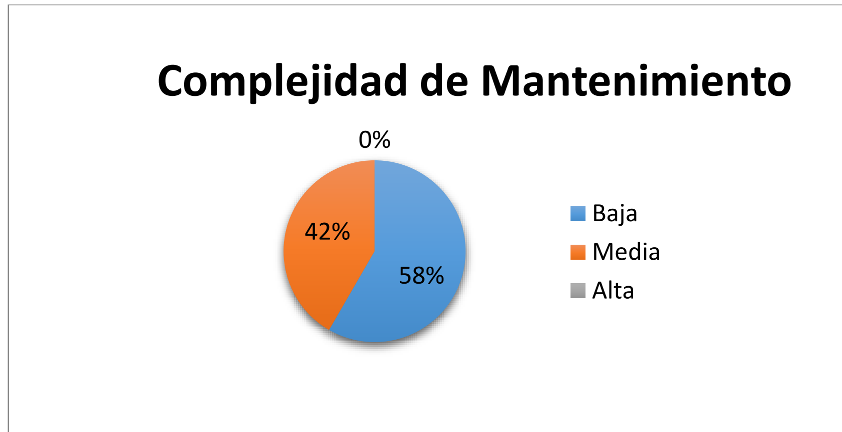

Figura 19: Grado de afectación del atributo Complejidad de Mantenimiento.

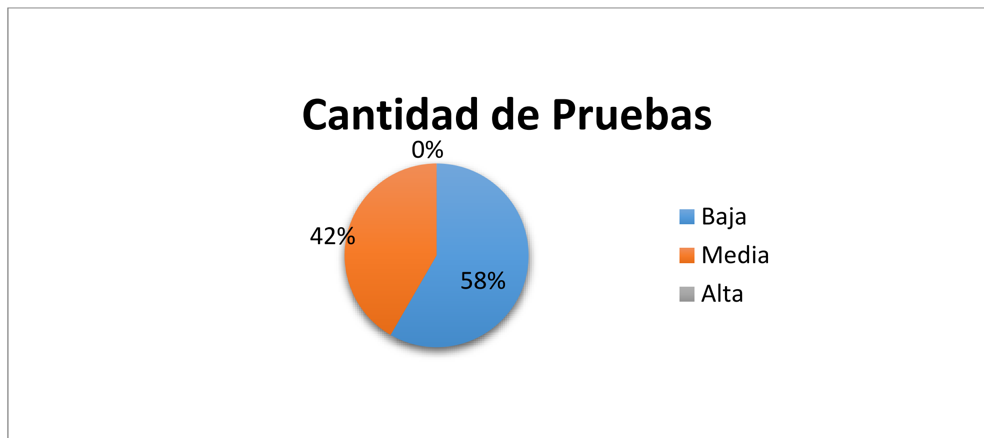

Figura 20: Grado de afectación del atributo Cantidad de Pruebas.

Acoplamiento: Según los resultados que se muestran, el 58% de las clases poseen un bajo acoplamiento, validando una realización correcta del diseño.

Complejidad de mantenimiento: Según los resultados que se muestran, el 58% de las clases se comportan de forma satisfactoria pues son de fácil soporte.

Cantidad de pruebas: luego de aplicar la métrica se obtuvo que el 58% de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones y pruebas de software.

Según lo analizado anteriormente, los valores de RC se comportan de forma satisfactoria siendo discretos en la mayoría de las clases, lo cual implica una disminución del acoplamiento y mayor facilidad de mantenimiento de las mismas, además de ser factible el diseño realizado.

3.3. Estrategia de pruebas

La estrategia es la guía que rige el proceso de pruebas de una aplicación enfocada en la verificación y validación de la propuesta de solución. A partir de la utilización del tipo de prueba funcional se definirá la estrategia de prueba a seguir. Para la verificación se realizarán pruebas con el método de prueba, caja blanca, para este método se usará la técnica de recorrido, camino básico, y se utilizará el método de caja negra para el acta de liberación otorgada por el grupo de calidad de CEGEL, formando parte de este proceso de verificación. Para la validación se realizarán pruebas utilizando el método de caja negra, como la carta de aceptación del cliente y la validación de la variable del problema.

3.4. Verificación de la propuesta

Para la verificación del sistema se realizan pruebas de unidad con el objetivo de comprobar el correcto funcionamiento del software, a nivel de codificación mediante el método de prueba de caja blanca respectivamente.

Pruebas de caja blanca

Las pruebas de caja blanca se basan en el conocimiento de la lógica interna del código del sistema, contemplan los distintos caminos que se pueden generar teniendo en cuenta las estructuras condicionales o los distintos estados del mismo. Para poner en práctica este tipo de pruebas se procede a utilizar la técnica del camino básico.

Técnica del camino básico

La técnica del camino básico es una técnica de prueba unitaria que permite al equipo de desarrollo obtener una medida de la complejidad lógica del código implementado como resultado del sistema propuesto en la investigación; usando dicha medida como guía para la definición de un conjunto de caminos independientes de ejecución, lo que garantiza que durante la prueba se ejecuten por lo menos una vez cada sentencia del programa (Pressman, 2005).

Complejidad Ciclomática

La complejidad ciclomática está basada en la teoría de grafos y da una métrica del software extremadamente útil, calculándose de tres formas:

- El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
- La complejidad ciclomática, V (G), de un grafo de flujo G se define como V (G)=A-N+2, donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.
- La complejidad ciclomática, V (G), de un grafo de flujo G también se define como V (G)= P+1, donde P es el número de nodos predicados contenidos en el grafo de flujo G.

En la Figura 21 se puede apreciar el código correspondiente al método editModuloAction perteneciente a la clase DocumentoErsController, se escogió este método debido a la relevancia que posee en el sistema propuesto. A continuación se procede a realizar el cálculo de la complejidad ciclomática.

- Confeccionar el grafo de flujo correspondiente.
- Calcular la complejidad ciclomática asociada.
- Extracción de los caminos independientes según el valor de la complejidad ciclomática.
- Realizar los casos de pruebas.
- Analizar los resultados obtenidos en las pruebas.

```
public function editModuloAction(Request $request, $id){
    $entity = $this->getModel()->getRepository()->find($id);//1
    if (!$entity) {//2
        throw $this->createNotFoundException('No se ha podido encontrar la entidad Modulo.');//3
    }

    $form = $this->createForm('documentoErs_form', $entity);//4
    $deleteForm = $this->createDeleteForm($id);//4
    unset($form['version'], $form['introduccion'], $form['proyecto']);//4
    $form->handleRequest($request);//4

    if ($form->isValid()) {//5

        $this->get('modulo_model')->persistArray($entity->getModulos(), $entity);//6
        $this->getModel()->save($entity);//6
        return $this->redirect($this->generateUrl('documentoers'));//6
    }

    return array(//7
        'entity' => $entity,//7
        'form' => $form->createView(),//7
    );
}
```

Figura 21: Código fuente que genera la funcionalidad modificar módulo.

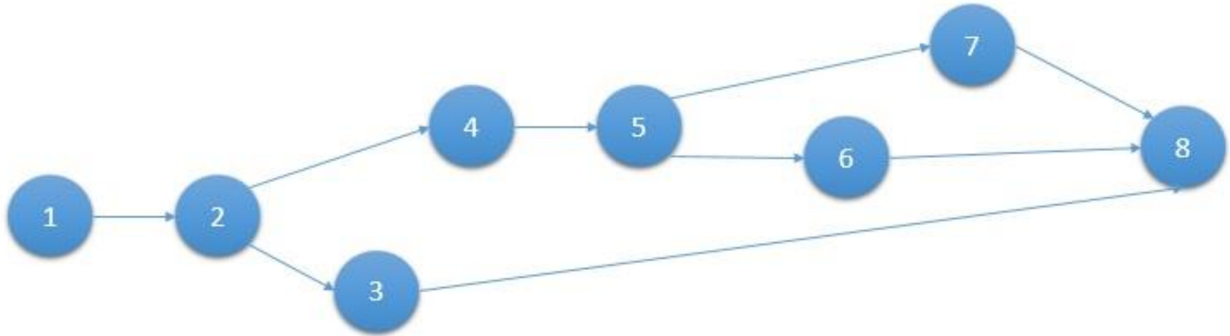

Figura 22: Grafo de flujo.

Cálculo de la Complejidad Ciclomática:

El número de regiones del grafo de flujo coincide con la complejidad ciclomática.

El número de regiones del grafo de flujo definido es 3 lo que deviene que se obtenga el siguiente resultado:

V (G) = 3

V (G) = (A-N)+2 = 9-8+2 = 3

V (G) = P+1 = 2+1 = 3

Una vez calculada la complejidad ciclomática por cada una de sus variantes se define como límite superior tres pruebas a realizar para que todo el código se encuentre probado.

Tabla 5: Caminos independientes establecidos.

Números	Caminos Básicos
1	1-2-3-8
2	1-2-4-5-6-8
3	1-2-4-5-7-8

Cada camino es un caso de prueba, de forma tal que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino como se refleja a continuación.

Caso de Prueba para el camino básico #1

$entity = ‘Null’, $id = 4

Condiciones de ejecución:

$entity = false

Resultados esperados: El sistema muestra un mensaje de error.

Resultados obtenidos: Satisfactorio

Caso de Prueba para el camino básico #2

$entity = ‘Documento’, $id = 6

Condiciones de ejecución:

$entity = true

```
$form[] = ‘documentoErs_form’, ‘Documento’
$form = $this->createForm(‘documentoErs_form’, $entity);
$form->handleRequest($request); = true;
```

```
$form -> isValid() = true
$this->get('modulo_model')->persistArray($entity->getModulos(), $entity);
$this->getModel()->save($entity);
```

Resultados esperados: El sistema muestra la vista de documentos con el módulo modificado.

Resultados obtenidos: Satisfactorio

Caso de Prueba para el camino básico #3

$entity = ‘Documento’, $id = 6

Condiciones de ejecución:

$entity = true

```
$form[] = ‘documentoErs_form’, ‘Documento’
$form = $this->createForm(‘documentoErs_form’, $entity);
$form->handleRequest($request); = false;
```

Resultados esperados: El sistema muestra un mensaje de error.

Resultados obtenidos: Satisfactorio

Pruebas de liberación

Las pruebas de liberación son destinadas a evaluar si al final de una iteración se cumplió la funcionalidad requerida por el cliente final. Estas pruebas se realizan a través del método de prueba de caja negra y aseguran el comportamiento del sistema y especifican los aspectos a probar cuando una historia de usuario ha sido correctamente implementada.

Para las pruebas de liberación se entregó la aplicación al grupo de calidad del centro CEGEL para hacerles pruebas funcionales. El proceso consta de 3 iteraciones, a continuación, se presentan los resultados obtenidos.

Tabla 6: No conformidades detectadas en las pruebas de liberación.

Iteraciones	No conformidades detectadas	No conformidades resueltas
1	29	29
2	0	0

Luego de realizar las 2 iteraciones se entregó el acta de liberación aseverando que la aplicación cumplía los requisitos mínimos para ser utilizada.

3.5. Validación de la propuesta

Con la validación de la propuesta se pretende comprobar que el software cumple las expectativas que el cliente espera. Como parte de la misma se valida a través de las pruebas de aceptación y la validación de la variable del problema planteado.

Pruebas de aceptación del cliente

Las pruebas de aceptación son destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente final. Estas pruebas aseguran el comportamiento del sistema y especifican los aspectos a probar cuando una historia de usuario ha sido correctamente implementada.
Se realizaron pruebas funcionales con el cliente en un entorno controlado, lo más similar posible al entorno real y los resultados obtenidos fueron satisfactorios.

Validación de la variable del problema

Es necesario validar la propuesta para comprobar el éxito de la investigación, para cumplir con este objetivo se aplicará un caso de estudio, así se comprobará la validez, objetividad y lo viable que puede ser la solución, para dar cumplimiento al objetivo general.

Con el desarrollo del sistema Gestión de Casos de Pruebas (GECAP), se facilita el Diseño de los Casos de Pruebas en los proyectos del Centro CEGEL ya que disminuye el tiempo que se tarda en realizar esta actividad, además de disminuir las No Conformidades (NC) que se detectan en los mismos, esto se evidencia a través de un caso de estudio realizado donde se tomó como referencia los Diseños Casos de Pruebas del proyecto Sistema de Gestión de Bufete Colectivo del centro CEGEL, ya revisados por el Grupo de Calidad de la Facultad con sus respectivas NC y los Diseños de Casos de Pruebas de ese mismo proyecto generados con la aplicación arrojando los siguientes resultados:

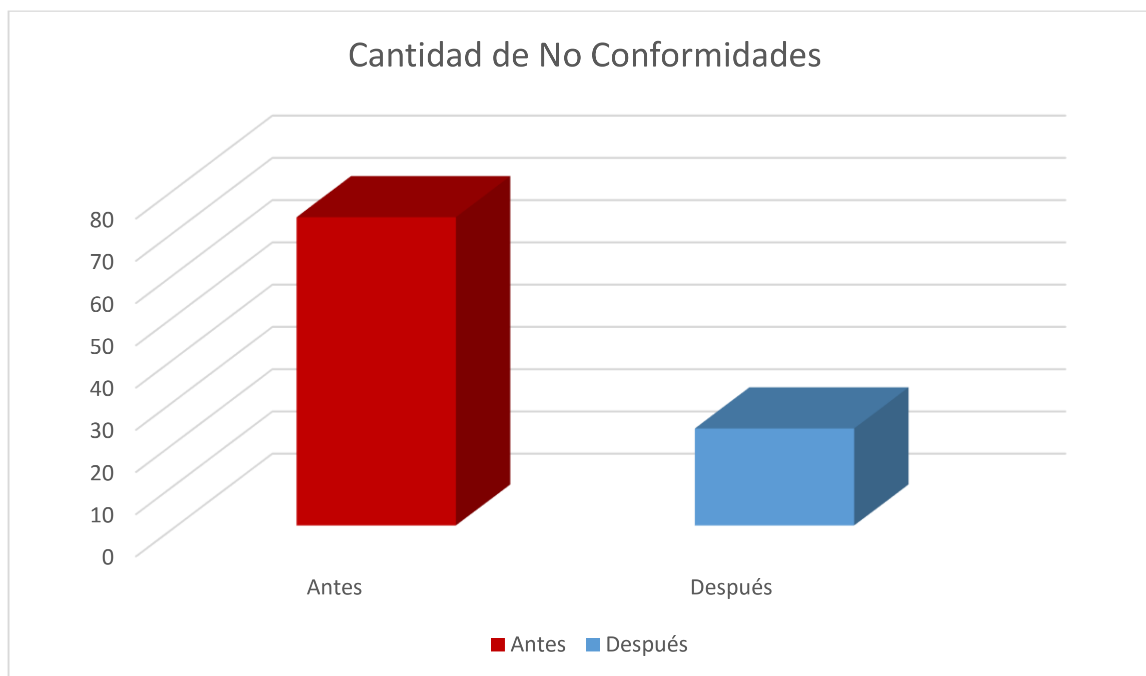

Figura 23: Cantidad de NC.

La gráfica anterior muestra la cantidad de NC detectadas en los Diseños de Casos de Prueba, el color rojo muestra la cantidad de NC detectadas antes de ser desarrollada la aplicación con un valor de 73 NC y el color azul las NC detectadas luego de generar los Casos de Pruebas con la aplicación con un valor de 23 NC, llegando a la conclusión de que se redujo considerablemente el número de NC detectadas.

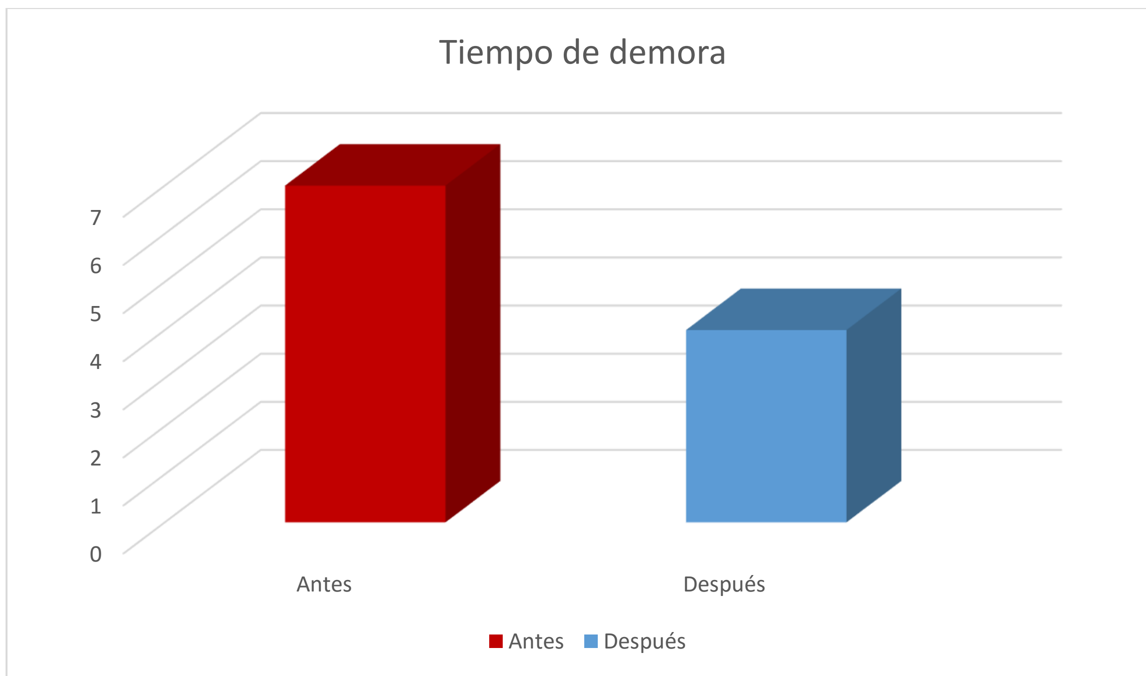

Figura 24: Tiempo de demora.

La gráfica anterior muestra el tiempo de demora en elaborar los Diseños de Casos de Prueba, el color rojo muestra el tiempo de demora antes de ser desarrollada la aplicación con un valor de 7 días y el color azul el tiempo de demora luego de generar los Casos de Pruebas con la aplicación con un valor de 4 días, llegando a la conclusión de que se redujo considerablemente el tiempo de demora.

3.6. Conclusiones parciales

A lo largo del capítulo se analizaron y construyeron los elementos imprescindibles que forman parte de la validación concluyendo lo siguiente:

Al aplicar la métrica TOC se obtuvieron valores positivos para los atributos de calidad responsabilidad, complejidad y reutilización.

Por otra parte, la aplicación de la métrica Relaciones entre Clases mostró un bajo acoplamiento entre las clases debido a la baja dependencia que existe entre las mismas además de revelar que la complejidad para realizar mantenimiento al código de la aplicación es baja y que la cantidad de pruebas a realizar para que el diseño de las clases sea óptimo será baja.

Los valores obtenidos en las 2 iteraciones de pruebas funcionales comprobaron que las funcionalidades descritas por el cliente se encontraban correctas.

Por último, se validaron las variables que forman parte del problema de la investigación, demostrando que con el sistema desarrollado se cumplen los objetivos planteados.

CONCLUSIONES

La elaboración del marco teórico de la investigación permitió adoptar las buenas prácticas que guían el desarrollo de software a nivel mundial.

La caracterización del proceso de generación de casos de pruebas basados en técnicas de pruebas funcionales permitió identificar a las técnicas de pruebas Partición de Equivalencia y Tabla de Decisión como medios para la generación de casos de pruebas funcionales a partir de la especificación de requisitos de software.

La utilización de las buenas prácticas y tecnologías seleccionadas permitió implementar una aplicación con un enfoque de calidad.

La aplicación de un caso de estudio permitió demostrar que en efecto el sistema genera casos de prueba a partir del documento de especificación de requisitos de software en un menor tiempo y con menor cantidad de NC.

RECOMENDACIONES

- Utilizar la herramienta en el proceso de desarrollo de software.
- Integrar la herramienta con un sistema de gestión de pruebas para ampliar las potencialidades del proceso.
- Agregar nuevas funcionalidades asociadas a otros tipos de pruebas en la herramienta que permitan extender su alcance.

BIBLIOGRAFÍA

9126-2, ISO/IEC. *Software engineering - Product quality - Part 2: External metrics.* s.l. : Final editorial correction version of Approved DTR Balloted 7N2419 in 2001 for ISO/IEC publish.

A Practical Approach to Validating and Testing Software Systems Using Scenarios. **J.Ryser, M. Glinz. 1999.** Bruselas : QWE '99, 3 rd International Software Quality Week Europe, 1999.

Aguilera, Serguio. 2010. Tipos de Software. 2010.

Alvarez, Miguel Angel. 2009. DesarrolloWeb.com. [Online] Marzo 9, 2009. http://www.desarrolloweb.com.

Calleja, Manuel Arias. 2009. *Etándares de codificación.* 2009.

Carolina Universidad Alejandro de Humboldt. 2009. SlideShare. [Online] abril 3, 2009. http://es.slideshare.net/Rubiano/automatizacion-de-pruebas-de-software-1245969.

Heumann, Jim. 2001. *Generating Test Cases From Use Cases.* s.l. : Rational software, 2001.

Ibrahim. 2007. *An Automatic Tool for Generating Test Cases form the System Requirements.* 2007.

jetbrains. 2016. www.jetbrains.com. [Online] 2016.

José, Ramírez Quintana, Orallo, José Hernández and César, Ferri Ramírez. 2004. *Introducción a la Minería de Datos.* Madrid : Pearson Educación, 2004.

Larman, Craig. 1999. *Introducción al análisis y diseño orientado a objetos.* 1999.

Martínez, Ramón Alexander Anglada. 2013. http://scielo.sld.cu/. [Online] 2013.

Matthias Riebisch, Ilka Philippow, Marco Götze. 2003. *UML-Based Statistical Test Case Generation.* Heidelberg, Alemania : Springer Verlag, 2003.

2007. *Mientras más entradas se tengan, más interesante se vuelve esta técnica, para los cuales se contemplarán datos de pruebas diversos, relacionados entre sí, pues ellos generarán un mayor número de combinaciones para probar.* 2007.

Mohamed Mussa, Samir Ouchani, Waseem Al Sammane, Abdelwahab Hamou-Lhadj. 2016. A survey of Model-Driven Testing Techniques. 2016.

Mora, Juan Tahuiton. 2011. *Arquitectura de software para aplicaciones Web.* 2011.

MyAdmin. 2016. www.phpmyadmin.net. [Online] 2016.

Noriega, Raúl. 2015. El proceso de desarrollo de software. 2015.

Palacio, Liliana González. 2009. *Método para generar casos de prueba funcional en el desarrollo de software.* Medellín, Colombia : Revista Ingenierías Universidad de Medellín, 2009.

PHP. 2016. http://www.php.com/. [Online] PHP, 2016.

Pressman, Roger S. 2005. *Ingeniería de Software- Un enfoque práctico.* 6ta. New York : MacGraw-Hill Education, 2005. ISBN 970-10-5473-3.

Rueda, Ana López-Mancisidor. 2013. *¿Por qué invertir en la automatización de pruebas Software?* España : Rational Software, 2013.

Rumbaugh. 2000. 2000.

Sánchez, Tamara Rodríguez. 06/03/2015. *Metodología de desarrollo para la actividad productiva de la UCI.* Habana : s.n., 06/03/2015.

Sensiolabs. 2016. http://twig.sensiolabs.org/. [Online] 2016.

Sommerville, Ian. 2011. *INGENIERÍA DE SOFTWARE.* Mexico : Pearson Educación, 2011. ISBN: 978-607-32-0603-7.

Symfony. 2016. http://symfony.es/. [Online] 2016.
TABLADEC 2009. **Castilla, María Josefina. 2009.** 2009.
Tortoisegit. 2016. https://tortoisegit.org/. [Online] 2016.
Trondheim, Norway. 2011. *Empirical studies of agile software development.* 2011.
Valdéz, José Luis Cendejas. 2014. 2014.
Vázquez, Yanetzi Jimeno and Rodríguez, Raúl M. Romero. 20013. *Herramienta para la generación automática de diseños de casos de prueba.* La Habana : s.n., 20013.
VP-UML. 2016. visual-paradigm.com. [Online] visual-paradigm, 2016.
Xetid. 2012. www.xetid.cu. [Online] 2012.