Universidad de las Ciencias Informáticas

Facultad 6

Título: "Sistema Informático para la Gestión de No Conformidades y Recursos del grupo de calidad de la facultad 6"

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

Autoras: Dairys Febles Pérez.
Daimelys Piloto Garaboto.

Tutores: Ing. Roig Calzadilla Díaz.
Ing. Diana Monné Roque.

Ciudad de la Habana, Junio 2010.
"Año 52 de la Revolución"

"…Este país vivirá de la inteligencia y de las producciones intelectuales…"

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Daimelys Piloto Garaboto

_____

Firma del Autor

Dairys Febles Pérez

_____

Firma del Autor

Ing. Roig Calzadilla Díaz

_____

Firma del Tutor

Ing. Diana Monné Roque

_____

Firma del Tutor

Datos de Contacto

DIPLOMANTES

Nombre: Daimelys Piloto Garaboto
E-mail: dpiloto@estudiantes.uci.cu

Nombre: Dairys Febles Pérez
E-mail: dfebles@estudiantes.uci.cu

TUTORES

Nombre: Roig Calzadilla Díaz
E-mail: rcalzadilla@uci.cu

Nombre: Diana Monné Roque
E-mail: dmonne@uci.cu

Agradecimientos

Son incontables las personas a las cuales debo agradecer el haber llegado hasta aquí: profesores de ayer y hoy, amigos de ayer y hoy, estando siempre en las buenas y malas junto a mí, animándome a seguir adelante ante cada nuevo reto que la vida me ha presentado, a todos los que un día me dijeron cuenta conmigo , muchísimas gracias pero especialmente a :

A las dos personas más importantes en mi vida, mi mamá y mi papá, los cuales desde el día en que nací dejaron de vivir para ellos y comenzaron a vivir para mí, siempre han abierto sus brazos cuando necesito un abrazo.

A mis abuelos lela, lelo y mima que siempre me han brindado su apoyo cuando más lo necesito, gracias por confiar en mí y estar seguros de que este momento llegaría, y a pipo, que aunque no estés físicamente yo sé que va a estar muy orgulloso porque siempre quisiste que yo fuera una profesional, no fui doctora, pero soy una ingeniera.

A mis tíos Daneris, Orlandito, Nena, José, Gracielita, Titica y Tomasito, por acogerme siempre como una hija más.

A todos mis primos por su apoyo en todo momento.

A Yenier por darme la fuerza que tanto hace falta en los momentos de debilidad, por estar siempre pendiente de mí, por quererme tanto y dejarme ocupar un lugar en su vida.

A mis amigas Yane y Daimelys por ser las hermanas que nunca tuve y brindarme tanto cariño en estos años y soportar mis risas y llantos.

A Lien y a Edilio que me han recibido en su casa con los brazos abiertos brindándome siempre su amor y cariño.

A mis tutores Diana y Roig por soportarnos todo este tiempo y además por su valiosa cooperación y su enorme dedicación.

Al tribunal que se ha portado genial en cada corte: critico y objetivo siempre.

A todas las personas que de una forma u otra contribuyeron a mi formación profesional y al desarrollo de este trabajo: Randy, Asdrúbal, Francisco, Javier, David, Norlen, Alberto, Hermes.

A Fidel Castro y a la Revolución por darme la posibilidad de convertirme en una profesional y a la Universidad de las Ciencias Informáticas por acogerme durante estos 5 años.

Dairys Febles Pérez

No será fácil en tampoco espacio dejar plasmado el nombre de todas las personas a las que les quiero agradecer por haber contribuido de una forma u otra con mi formación como ingeniera. Les agradezco de todo corazón:

A mis padres, que han guiado mi transitar por la vida, gracias por el apoyo y el ánimo que me han brindado, sin ustedes hubiera sido imposible realizar este trabajo, gracias por confiar en mí, los quiero mucho.

A mis abuelos Zunilda, Félix, Lourdes, Víctor, María y Orlando, en especial a pipo y mima, que me dedicaron toda su vida y amor y que sé que para ellos este será uno de los momentos más felices de sus vidas.

A mis hermanitos, tíos y primos por su apoyo y cariño en todo momento.

A Delvis, por el apoyo diario, por extenderme su mano y estar a mi lado en los momentos más difíciles, por escuchar mis lamentos, por estar siempre pendiente de mi, por comprenderme, soportarme y quererme, muchas gracias por todos los momentos de felicidad.

A mi compañera de tesis por ser la mejor amiga del mundo, por tantas experiencias vividas en estos cinco años, por su confianza y sus enseñanzas, por hacer realidad juntas este gran sueño.

A Yane por soportar mis majaderías, por escucharme y decirme siempre las cosas como son. Gracias por tenerme siempre presente, por tu cariño infinito, en fin por ser una amiga maravillosa.

A mis tutores Roig y Diana por su guía, apoyo y experiencia profesional, la cual me ha enriquecido como futura profesional, por su exigencia y crítica en la elaboración de este trabajo.

A todas las personas que aportaron su granito de arena para que este trabajo saliera adelante: Randy, Francisco, Asdrúbal, David, Alberto, Javier, Norlen, Hermes, a todos muchas gracias.

A todos los profesores que han estado presentes en mi vida, que ahora pueden ver el fruto de su trabajo.

A nuestro Comandante en Jefe Fidel por tener la idea de crear esta gran universidad y darnos la oportunidad de formarnos en ella como profesionales.

Daimelys Piloto Garaboto

Dedicatoria

A la mujer más hermosa de este mundo, mi mamá, la principal causante que este momento se haya hecho realidad, por lograr este sueño que también es de ella, por guiar cada unos de mis pasos, por brindarme siempre su amor a cambio de nada, por ser la persona más especial de este mundo.

A mi eterno amor, mi papi, por su amor, sus enseñanzas, sus años de sacrificio, su confianza, sus consejos, y apoyo en cada momento, por confiar tanto en mí y estar seguro que no lo defraudaría, te quiero mucho.

A mis abuelos por el amor que me han brindado, por estar siempre en los momentos más difíciles, por apoyarme y estar siempre pendientes de mí.

A Yenier por estar siempre a mi lado, por darme tanto apoyo en el momento que más lo necesitaba, por comprenderme tanto.

Dairys Febles Pérez

A mi mamita linda que es lo más grande que tengo en la vida, en todo lo que hago siempre te tengo presente, y solo busco enorgullecerte cada día más, no tienes una doctora como deseabas pero si una ingeniera, te adoro.

A mi papi por ser el mejor padre que hubiera podido tener, por la confianza que siempre has depositado, por ser mi guía, siguiendo tus pasos como ingeniero hoy cumplo realidad uno de mis mayores sueños, te quiero mucho.

A mi padrastro por quererme como a una hija, por sus consejos, por su dedicación, por estar siempre pendiente de mí.

A mis abuelos, especialmente a mi pipo y mi mima por su incansable preocupación por mi futuro, por quererme tanto y estar siempre pendiente de mis pasos.

A mis hermanitos Daily, Lía y Danito que son mis tesoros más pequeños, sin ustedes la felicidad no fuese completa, gracias por existir.

A Delvis por su dedicación y esmero conmigo, por su paciencia y comprensión, gracias por permitir que sea una parte pequeñita de tu vida, te amo.

Daimelys Piloto Garaboto

Resumen

La investigación está asociada al trabajo del grupo de calidad de la facultad 6, que se dedica a probar cada software que se elabora en la misma con el objetivo de detectar la mayor cantidad de no conformidades posibles y de esta forma asegurar la calidad del mismo. Para dar respuesta a la siguiente interrogante: ¿Cómo mejorar los procesos de Gestión de No Conformidades y Recursos en el grupo de Calidad de la facultad 6? se desarrolló una herramienta informática para automatizar los procesos de Gestión de No Conformidades y Recursos, diseñando una aplicación web capaz de tramitar todos los elementos de las No Conformidades y los Recursos. Esta aplicación permitirá al Asesor de Calidad y al personal autorizado, interactuar a través de la red con la información almacenada en la aplicación, así como hacer reportes de seguimiento para llevar un control minucioso de todo lo referente a las pruebas de software. Para garantizar la portabilidad de la aplicación se desarrolló bajo ambientes multiplataforma, fundamentalmente con herramientas de software libre.

Palabras Claves

Calidad de Software, No Conformidades, Recursos.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS.

Introducción

Una de las principales características de la naturaleza del hombre (como género) es la inconformidad. Esta característica nos ha llevado a ser perfeccionistas, a tratar de lograr cada vez que se plantea resolver un problema, resolverlo de la mejor forma. Es ahí donde gracias a la inconformidad, surge la calidad, en busca de soluciones que verdaderamente cumplan con las expectativas.

La calidad en el desarrollo del software se ha convertido actualmente en uno de los principales objetivos estratégicos de las entidades que se dedican a esta producción, debido a que cada vez más los procesos principales de dichas empresas dependen de los sistemas informáticos para su buen funcionamiento (1).

La calidad del software está determinada por el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. Calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. Ésta es medible y varía de un sistema a otro. Existen varios modelos y normas que contribuyen a aumentar la calidad del proceso de desarrollo de software, tales como: Capability Maturity Model Integration (CMMI), Organización Internacional para la Estandarización (ISO).

El desarrollo acelerado de la ciencia y las tecnologías de la información, así como la velocidad de cambio en el manejo de los negocios, ha traído como consecuencia que las empresas informáticas enfrenten cada día un reto para brindar una respuesta rápida, eficaz y con calidad a los clientes, que cada vez se vuelven más exigentes, no sólo en cuanto al precio sino también en la confiabilidad que deben brindar los productos de software.

Con el crecimiento de esta industria en Cuba, el tema de la calidad de productos informáticos se hace obligatorio en empresas que comercializan estas aplicaciones. Un ejemplo de estas entidades en Cuba son: DeSoft, Softel y la Universidad de las Ciencias Informáticas, que a pesar de ser un centro destinado a formar profesionales altamente calificados en esta rama, produce software y ofrece servicios informáticos. Con la creación de esta universidad y paralelo a esto el aumento de la producción de software, la necesidad de fabricar un producto con calidad creció, esto fue la base para el surgimiento de un laboratorio de certificación de pruebas de software y junto a este los grupos de calidad en cada una de las facultades, encargados de asegurar la calidad en sus proyectos productivos, donde los estudiantes desempeñan un rol protagónico.

Con la observación de la práctica diaria, el intercambio con especialistas y estudiantes relacionados con el tema y la experiencia adquirida en el desempeño pre-profesional se pudieron detectar algunas irregularidades en el manejo de las informaciones en el laboratorio de calidad de la facultad 6, tales como:

- La accesibilidad por parte de los involucrados a la información de las No Conformidades es insuficiente.
- No se encuentra disponible un control sistemático del trabajo realizado por cada uno de los estudiantes.
- No se encuentra organizado el control del estado real de los medios computarizados disponibles para realizar el trabajo.
- No existen registros que muestren el seguimiento del trabajo realizado en cada una de las iteraciones en la revisión del producto.
- La información no se encuentra organizada de la mejor manera por lo que a la hora de buscar algunos datos se le hace muy engorroso al usuario.
- La gestión de los recursos no se hace de la forma más eficiente, pues no se tiene un control exacto de los recursos humanos y tecnológicos.

Teniendo en cuenta la situación problemática existente se plantea como ***problema científico*:**
¿Cómo mejorar los procesos de Gestión de No Conformidades y Recursos en el grupo de Calidad de la facultad 6?

Donde el ***objeto de estudio*** es: Los procesos de Gestión de Pruebas de Software.

Se estima como ***campo de acción*:** Los procesos de Gestión de No Conformidades y Recursos en el grupo de Calidad de la facultad 6.

Como ***objetivo general*** se tiene: Desarrollar un sistema informático para la Gestión de No Conformidades y Recursos en el grupo de Calidad de la facultad 6.

Desglosado en los sucesivos ***objetivos específicos*:**

1. Definir las funcionalidades que deberá cumplir el software.
2. Diseñar el software para la Gestión de No Conformidades y Recursos en el grupo de calidad de la Facultad 6.

3. Implementar una aplicación web a partir de los Procesos de Gestión de No Conformidades y Recursos en el grupo de calidad de la Facultad 6.
4. Probar la aplicación web desarrollada.

Para dar cumplimiento a los objetivos trazados, se plantean las siguientes ***tareas*:**

1. Estudio de los principales conceptos, herramientas, relacionados con los procesos de Gestión de No Conformidades y Recursos.
2. Estudio y selección de la metodología de desarrollo.
3. Estudio y selección de los lenguajes de modelado y de programación a utilizar.
4. Estudio y selección de las herramientas para desarrollar el software.
5. Estudio y selección del framework de desarrollo a utilizar.
6. Realización de entrevistas a especialistas de Calidad para el levantamiento de requisitos.
7. Modelado del negocio.
8. Especificación de los requerimientos funcionales y no funcionales del software.
9. Realización del análisis y diseño del software.
10. Implementación del software.
11. Realización de pruebas de caja negra.
12. Análisis de los resultados obtenidos.

El contenido de este documento está estructurado de la siguiente manera:

Capítulo 1: *Fundamentación Teórica:* Se explica el estado del arte, las metodologías, las herramientas y los lenguajes utilizados para el desarrollo de la aplicación.

Capítulo 2: *Características del Sistema:* Se define el Modelo de Negocio, Especificación de los Requisitos del Sistema y Casos de Uso.

Capítulo 3: *Análisis y Diseño del sistema:* Se analizan los Casos de Uso del Sistema para diseñar las clases que se implementarán, se representan los Diagramas de Secuencia del Diseño, el Diagrama de las Clases diseñadas con sus relaciones, los principios utilizados para el diseño de dichas clases, el Diagrama de Clases Persistentes, el Diagrama Entidad Relación y la descripción de las tablas de la Base de Datos.

Capítulo 4: *Implementación y Prueba:* Está enfocado a la implementación del sistema con el objetivo de darle solución a los requisitos funcionales. Se realiza el Modelo de Implementación y se aplican patrones de diseño y de arquitectura a cada una de las clases del diseño. Además de probar todas las funcionalidades del software.

Capítulo 1: Fundamentación Teórica.

En este capítulo se estudia el estado actual de los software de gestión de no conformidades y seguimiento de los recursos. Se hace énfasis en las comparaciones de las Metodologías, Lenguajes de Programación, IDE de Desarrollo, Gestor de Base de Datos, Framework, para luego realizar la selección de lo que se utilizará en el desarrollo del software.

1.1 Calidad de Software.

La calidad del software es una preocupación a la que se dedican muchos esfuerzos. Sin embargo, el software casi nunca es perfecto. Todo proyecto tiene como objetivo producir software de la mejor calidad posible, que cumpla, y si puede supere las expectativas de los usuarios.

La creciente necesidad de desarrollar software complejos en cortos períodos de tiempo, con menores esfuerzos, ha favorecido el crecimiento de una cultura que aboga por realizar los productos cada vez con mayor calidad, esto ha provocado que en los últimos tiempos este término sea frecuentemente utilizado y aparezca en multitud de contextos, pues con él se busca despertar en quien lo escucha una sensación positiva, trasmitiendo la idea de que algo es mejor, es decir, la idea de excelencia.

Existen varias definiciones de la Calidad del Software expresadas por diversas instituciones y personalidades destacadas en este sector, tales como:

- “Conjunto de características de un producto o servicio que le confieren su aptitud para satisfacer necesidades expresadas e implícitas”. (2)
- “La calidad del software es el grado con el que un sistema componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario”. (3)
- “Concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario”. (4)

1.1.1 Pruebas de Software.

Para contribuir con la calidad del software es recomendable que el producto software vaya siendo evaluado a medida que se va construyendo. Posteriormente se hace necesario llevar a cabo paralelo al

proceso de desarrollo, un proceso de evaluación y comprobación de los distintos productos o modelos que se van generando, en el que participarán desarrolladores y clientes. Las pruebas de software constituyen un pilar indispensable para evaluar y determinar la calidad de un software. Concretamente se puede definir pruebas de software como:

- Una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. (5)

1.1.2 No Conformidades.

Las pruebas de software no garantizan que un software esté libre de errores, sino que se detecten la mayor cantidad de no conformidades posibles en el mismo para su debida corrección. De acuerdo a la norma ISO 9000 una no conformidad es el incumplimiento de un requisito. (6)

1.2 Sistemas automatizados para la Gestión de No Conformidades.

1.2.1 SIGNO – Software para la Gestión de No Conformidades.

Este sistema es desarrollado en Colombia en el año 2005. Es una herramienta informática de gestión que permite hacer el registro y la gestión total de las no conformidades, tanto internas como externas, para un sistema de gestión de la calidad ISO 9000:2000. SIGNO permite registrar toda la información de una no conformidad, asignar responsables (automática o manualmente), reportar la respuesta, reportar el problema y sus causas, definir el plan de acción (acciones correctivas/preventivas), visualizar mediante semáforos la posible demora en ejecutar acciones, reportar la ejecución de acciones, reportar la auditoría realizada, cerrar no conformidades y realizar consultas específicas por diferentes criterios. Adicionalmente permite todo el manejo de usuarios y motivos para hacer una fácil adaptación a las necesidades de su organización.

Limitaciones:

El costo, que va más allá de las posibilidades, el cual asciende a 100 000 euros, además de no ser configurable a la hora de decidir qué pruebas hacer. Las pruebas realizadas en la Universidad tienen características específicas a la hora de hacer alguna revisión en dependencia del cronograma de trabajo y las necesidades del cliente.

1.2.2 KMKey Quality.

KMKey Quality es un software de gestión de calidad ideal para la implantación y mantenimiento de un sistema de gestión de calidad (SGC) de cualquier tipo: ISO 9001, ISO 14001, entre otros, o de una combinación de los mismos, facilitando su integración. Mediante KMKey Quality podrá gestionar y mantener la documentación del sistema, los registros, y los flujos de información propios que se generan en acciones como la gestión de No Conformidades, Acciones Correctivas/Preventivas, Reclamaciones, Auditorías, Indicadores, Evaluaciones, entre otros. Todo ello adaptado al enfoque propio de su organización.

Dentro de sus funcionalidades se encuentra:

No conformidades y acciones correctivas/preventivas:
Desde el mismo gestor de expedientes cualquier usuario autorizado puede introducir No Conformidades, así como planificar las correspondientes Acciones Correctivas o Preventivas, repartiendo las tareas a cada uno de los responsables, y controlando plazos y acciones realizadas.

Limitaciones:
El software KMKey Quality dentro de sus funcionalidades se encuentra la Gestión Documental, la gestión de expedientes específicos para Planes de Formación, Planes de Auditorías o Planes de Mejora Continua, así como el Seguimiento y Evaluación de Proveedores o la Gestión y Control de Indicadores, a pesar de ser un software libre estas funcionalidades están todas correlacionadas, por lo que este software no se ajusta a las necesidades de la gestión de no conformidades en el grupo de calidad.

Las herramientas descritas anteriormente no satisfacen las necesidades existentes en el laboratorio de calidad de la facultad 6, de ahí surge la necesidad de crear el sistema informático para la gestión de no conformidades y recursos, la cual será desarrollada para facilitar el trabajo del grupo de calidad de la facultad, dentro de sus funcionalidades se encuentra la gestión de no conformidades y recursos.

1.3 Metodologías de Desarrollo.

Una Metodología de desarrollo es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un producto de software. Una metodología representa el camino para desarrollar software de una manera sistémica.

1.3.1 Rational Unified Process (RUP).

El Proceso Unificado RUP, constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos. Proporciona una aproximación disciplinada a la asignación de tareas y responsabilidades. RUP actúa como modelo y puede ser adaptado y extendido. (7)

El ciclo de vida de RUP se caracteriza por:

- Dirigido por casos de uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).
- Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los casos de uso relevantes desde el punto de vista de la arquitectura.
- Iterativo e Incremental: RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son mini proyectos.

Fases:

- Concepción o inicio: Comprender los requisitos y determinar visión y alcance del proyecto.
- Elaboración: Asignar recursos, especificar las características y definir la arquitectura.
- Construcción: Implementación, construir el producto operacional.
- Transición: Hacerlo operativo para los usuarios, nivel correcto de calidad para entregar.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo.

Flujos de trabajo:

- Modelamiento del negocio: Describe los procesos de negocio, identificando quienes participan y las actividades que requieren automatización.
- Requerimientos: Define que es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y diseño: Describe como el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: Define como se organizan las clases y objetos en componentes, cuales nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Prueba (Testeo): Busca los defectos a lo largo del ciclo de vida.
- Instalación: Produce la liberación del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.
- Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- Administración de configuración y cambios: Describe como controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.
- Ambiente: Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

RUP se basa mucho en la documentación, que aunque se ve afectada con los posibles cambios volátiles que los clientes soliciten en cuanto a funcionalidades del software, representa una gran ventaja, ya que gracias a su plan de desarrollo se pueden reconocer los problemas y fallos de forma temprana y corregirlos. Pero además, RUP no es un sistema con pasos firmemente establecidos, sino una metodología adaptable al contexto y necesidades.

1.3.2 eXtreme Programming (XP).

Programación Extrema (Extreme Programming, XP). Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.

Las principales características son:

- Centrado en resolver el problema lo más rápido posible.
- Cada miembro del equipo debe estar listo para enfrentar cualquier problema.
- El cliente se introduce en el equipo de desarrollo.
- Hago algo y lo pruebo.
- Termino todo y después integro.

Los obstáculos más comunes surgidos en proyectos XP son la "fantasía" de pretender que el cliente se quede en el sitio y la resistencia de muchos programadores a trabajar en pares.

1.3.3 OpenUP.

OpenUP es una versión más ágil de lo que es RUP. Éste nos plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo corto. Principalmente plantea que se debe utilizar solo los procesos que sean necesarios y en este caso se necesita de personas y profesionales que sean capaces de distinguir entre lo necesario y lo que no es necesario para que el proyecto no tenga errores y con eso evitar entregar al usuario final un producto de mala calidad, además plantea que no se deben utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario pudiendo ser modificado, mejorado y extendido.

1.3.4 Metodología Seleccionada.

Anteriormente se analizaron algunas metodologías como XP, que a pesar de ser muy ágil en el proceso de desarrollo de software debido a que utiliza el mínimo de documentación, tiene como desventaja que el cliente debe de estar fuertemente ligado al equipo de desarrollo, en este caso el cliente no se encuentra dentro del equipo de desarrollo.

Otra metodología estudiada fue OpenUP, la cual es muy simple en el proceso de desarrollo de software generando poca documentación, por lo que no se adapta a las necesidades del equipo de desarrollo.

Después de haber analizado estas tres metodologías de desarrollo de software se decidió utilizar RUP, ya que es la que más se adapta a las necesidades del equipo de desarrollo. Además constituye un marco de trabajo genérico que puede ser adaptado a una gran diversidad de sistemas de software independientemente del tamaño del proyecto, el tipo de organización y los diferentes niveles de aptitud.

1.4 Roles y Artefactos.

Un rol es una definición abstracta de un conjunto de responsabilidades para las actividades que deben ser realizadas y para los artefactos que se producirán.

En RUP se define los artefactos como productos tangibles del proyecto que son producidos, modificados y usados. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

RUP propone un conjunto de roles, agrupados por participación en tareas relacionadas como son: Analistas, Desarrolladores, Gestores, Apoyos, Especialista en Prueba y Otros Roles Adicionales. Cada uno es responsable de la realización de un grupo de artefactos en correspondencia con los flujos de trabajo.

De acuerdo con las necesidades y características de la investigación los roles que se desarrollarán serán analista, diseñador, diseñador de base de datos, probador y programador pertenecientes a los grupos de analistas y desarrolladores definido por el Proceso Unificado de Desarrollo (RUP) respectivamente.

Analista: Es el responsable del conjunto de requisitos que están modelados en los casos de usos específicos, es su responsabilidad también delimitar el sistema, encontrando los actores y los casos de uso y asegurando que el modelo de casos de uso es completo y consistente.

Los artefactos que serán elaborados por el analista serán los siguientes:

- Modelo de Caso de Uso: Se utiliza como acuerdo entre el cliente y desarrolladores, y proporciona la entrada fundamental para el análisis, el diseño y las pruebas.

- Glosario de Términos: Define los términos más importantes que se usan en el proyecto. Es muy útil para alcanzar un consenso entre los desarrolladores relativo a la definición de diversos conceptos y para reducir el riesgo de confusiones.
- Actor del Sistema: No es más que el entorno externo del sistema. En este artefacto quedan representados mediante uno o más actores cada uno de los tipos de usuarios y de los sistemas externos que interactúan con el sistema.
- Caso de Uso del Sistema: Es el artefacto que representa cada forma en que los actores usan el sistema.
- Vista de Caso de Uso: Incluye los casos de uso que describan alguna funcionalidad importante y crítica, es decir describe los casos de uso significativos para la arquitectura.

Diseñador de Sistema: Es el responsable del diseño de una parte del sistema, el cual contiene los requisitos, la arquitectura y el desarrollo de proceso para el proyecto. Además identifica y define las responsabilidades, operaciones, atributos y relaciones de los elementos de diseño.

A continuación se describen los artefactos que serán elaborados por el diseñador en el proyecto:

- Clase del Diseño: Se define como la descripción de un conjunto de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos y semántica.
- Realización de caso de uso: Describe como un caso de uso particular es realizado dentro del modelo de diseño, en términos de colaboración de objetos.
- Subsistema de Diseño: Encapsula el comportamiento, proporcionando interfaces explícitas y formales, sin exponer su contenido interno. Proporciona la capacidad de poder encapsular las interacciones de un número de clases y/o subsistemas. Representa una forma de organizar los artefactos del modelo de diseño en piezas más manejables.

Diseñador de Base de datos: Es el responsable de diseñar el almacenamiento de la información persistente que se usará en el proyecto. Debe definir el diseño detallado de la BD incluyendo tablas, índices, vistas, restricciones, procedimientos almacenados y cualquier otro elemento que se necesite para almacenar, recuperar y eliminar objetos persistentes.

Esta información se recoge en el artefacto Modelo de Datos:

- Modelo de datos: Describe la representación lógica y física de los datos persistentes usados por la aplicación. Puede ser inicialmente creado a través de ingeniería inversa de un almacenamiento de datos persistentes que ya exista (base de datos) o puede ser inicialmente creado a partir de un conjunto de clases del diseño persistentes en el modelo de diseño.

Implementador: Es el responsable de desarrollar y probar componentes, de acuerdo con las normas adoptadas en el proyecto para la integración de subsistemas más grandes.

El artefacto que elaborará el implementador se describe a continuación:

- Diagrama de componentes: Son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación, especificando los subsistemas de implementación y sus dependencias a la hora de importar código y organizar los subsistemas de implementación en capas.

Probador: Es el responsable durante las actividades principales de las pruebas, el cual incluye la conducción de las pruebas necesarias y el registro del resultado de la prueba.

El artefacto que elaborará el probador se describe a continuación:

- Diseño de Caso de Pruebas: Comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente. Un caso de prueba se diseña según las funcionalidades descritas en los casos de usos. Este diseño se elabora previamente a realizar las pruebas funcionales a la aplicación. Se parte de la descripción de los casos de usos del sistema, como apoyo para las revisiones. Cada planilla de caso de prueba recoge la especificación de un caso de uso, dividido en secciones y escenarios, detallando las funcionalidades descritas en él y describiendo cada variable que recoge el caso de uso en cuestión, además quedan plasmadas las revisiones realizadas al caso de prueba; así como un registro de todo aquello que no corresponde a la calidad del software. Existen casos de pruebas para los diferentes métodos: Caja Negra y Caja Blanca.

1.5 Lenguaje de Modelado.

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar (parte de) un diseño de software orientado a objetos.

1.5.1 UML.

Debido a la selección de la metodología de desarrollo de software se hace necesario utilizar como Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language).
UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Está pensado para poder aplicarse en cualquier medio que necesite capturar requerimientos y comportamientos del sistema que se desee construir. Ayuda a comprender y a mantener de una mejor forma un sistema basado en un área que el analista o desarrollador puede desconocer. UML surgió con el propósito de lograr una estandarización universal al crecido número de metodologías de desarrollo que habían estado apareciendo.

UML no es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso.

La decisión de utilizar UML como notación para el desarrollo del software se debe a que se ha convertido en un estándar que tiene las siguientes características:

- Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- UML es independiente del proceso, aunque para utilizarlo óptimamente se debería emplear en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

1.6 Herramienta de modelado.

Las herramientas de modelado de sistemas informáticos se emplean para la creación de modelos de sistemas que ya existen o que se desarrollarán.

Las buenas herramientas de modelado cumplen con determinadas características:

- Permiten una visión descendente del sistema.
- Permiten particionar el sistema.
- Poseen componentes gráficos con algo de apoyo textual.
- El modelo resultado debe ser fácil de comprender.
- Poseen mínima redundancia.

1.6.1 Visual Paradigm.

Visual Paradigm es una herramienta UML profesional fácil de usar. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Dentro de las principales características de la herramienta están:

- Soporte de UML versión 2.0.
- Disponibilidad de integrarse en los principales IDEs.
- Soporta una gama de lenguajes en la Generación de Código e Ingeniería Inversa en Java, C++, CORBA IDL, PHP, Ada y Python.
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio.
- Diagramas de flujo de datos.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas.
- Importación y exportación de ficheros.
- Editor de figuras.

Visual Paradigm ofrece un entorno amigable para el usuario, sugiere nuevos posibles componentes a utilizar, por lo que ya no es necesario localizarlos en la barra y así se crea fácilmente cualquier tipo de diagrama. Incluye gran variedad de estereotipos para la creación de diagramas de fácil entendimiento, los que organiza automáticamente.

1.6.2 Rational Rose.

Rational Rose se ha convertido en una herramienta de modelado visual para el análisis y diseño de sistemas basados en objetos. Con el uso del lenguaje común de modelado UML, facilita el trabajo para el equipo de desarrollo y garantiza mayor eficiencia y calidad del trabajo. Permite generar código en diferentes lenguajes de programación partiendo de modelado UML. Además, hace posible efectuar la ingeniería inversa, es decir, obtener información del diseño de un software partiendo de su código.

Rational Rose permite mantener la consistencia de los modelos del sistema software, realiza chequeo de la sintaxis UML y genera documentos automáticamente.

1.6.3 Herramienta Seleccionada.

Durante el estudio realizado sobre las herramientas de modelado se ha podido determinar que ambas poseen muchas ventajas a la hora de realizar el modelado de un software, sin embargo Rational Rose presenta como desventaja que es propietario, por lo que no sería recomendable utilizarlo para el desarrollo de la aplicación, pues en la actualidad se aboga por la soberanía tecnológica y la utilización de software gratuitos.

Visual Paradigm es la herramienta CASE que se utilizará en la modelación de este proyecto debido a que es multiplataforma, además de ser una potente herramienta CASE con licencia libre para uso de la comunidad de desarrolladores. Permite modelar UML, con 13 tipos diferentes de diagramas que pueden ser exportados como imágenes en formato JPG, PNG, entre otros. Proporciona a los desarrolladores una plataforma con interfaz amigable que les permite diseñar un producto con calidad de forma muy rápida. Puede ser extendido, pues presenta soporte al diseño personalizado, permitiendo incorporar nuevas formas y notaciones, mediante el uso de imágenes o íconos importados.

1.7 IDE de Desarrollo.

Un entorno de desarrollo integrado o IDE es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

1.7.1 Zend Studio.

Zend Studio consta de dos partes en las que se dividen las funcionalidades de parte del cliente y las del servidor. Las dos partes se instalan por separado, la del cliente contiene la interfaz de edición y la ayuda. Permite además hacer depuraciones simples de scripts, aunque para disfrutar de toda la potencia de la herramienta de depuración habrá que disponer de la parte del servidor, que instala Apache y el módulo PHP o, en caso de que estén instalados, los configura para trabajar juntos en depuración.

Zend Studio fue diseñado para usarse con el lenguaje PHP; sin embargo ofrece soporte básico para otros lenguajes Web, como HTML, Javascript y XML.

1.7.2 NetBeans 6.8 Beta.

El IDE NetBeans una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java y soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). NetBeans IDE 6.8 Beta proporciona herramientas para la programación en el lenguaje PHP e incluye resaltado de sintaxis, auto-completado de código, templates, documentación de PHP integrada, entre otras funcionalidades. Provee soporte para el framework Symphony y cuenta con un módulo de subversion para facilitar la manipulación de las versiones del código durante la implementación (8).

1.7.3 IDE Seleccionado.

Después de haber realizado esta comparación entre los IDE de Desarrollo se puede apreciar que ambos son potentes en el desarrollo de aplicaciones web, sin embargo el ZendStudio tiene como desventaja el costo, que se va más allá de las posibilidades, por lo que se decidió utilizar como IDE de Desarrollo NetBeans ya que es un reconocido entorno de desarrollo integrado disponible en múltiples plataformas,

permite desarrollar aplicaciones web con gran rapidez, además es un producto libre y gratuito sin restricciones de uso.

1.8 Lenguaje de Programación.

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina.

1.8.1 PHP 5.

PHP (Hypertext Preprocessor) es un lenguaje script para el desarrollo de páginas web dinámicas del lado del servidor, cuyos fragmentos de código se intercalan fácilmente en páginas HTML, debido a esto, y a que es de Open Source (código abierto), es el más popular y extendido en la web.

PHP tiene muchas ventajas algunas de ellas son:

- Multiplataforma: Inicialmente fue diseñado para entornos UNIX por lo que ofrece más prestaciones en este sistema operativo, pero es perfectamente compatible con Windows.
- Soporte para varios servidores Web.
- Mucha documentación (Ejemplos, manuales).
- Posee una sintaxis bastante clara.
- Fácil aprendizaje.
- Seguro.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Además PHP está orientado a objetos.

Por las potencialidades que ofrece, se decidió utilizar como lenguaje de programación PHP, debido a su amplia distribución está perfectamente soportado por una gran comunidad de desarrolladores. Como producto de código abierto goza de la ayuda de un gran grupo de programadores, permitiendo que los fallos de funcionamiento se encuentren y se reparen rápidamente. El código se pone al día continuamente con mejoras y extensiones de lenguaje para ampliar las capacidades de PHP.

1.9 Framework de Desarrollo.

Un framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un framework proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

1.9.1 Symphony.

Symphony 1.2 es un framework de PHP que simplifica el trabajo con páginas Web dado a la automatización de algunos patrones para resolver tareas frecuentes. Este framework tiene una estructura de código que fuerza al desarrollo de código más legible. También encapsula operaciones complejas en instrucciones sencillas. Symphony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones Web. Este framework trabaja con el patrón de arquitectura Modelo Vista Controlador y la herramienta Propel para el mapeo de objetos a bases de datos. Es un framework sencillo para cualquier programador independientemente de que sea principiante o un experto en PHP. Symphony está programado en PHP5 y enfocado al desarrollo en el mismo lenguaje. Este framework hace un uso completo de los mecanismos de POO (Programación Orientada a Objetos) disponibles en PHP5. (9)

Symphony separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.

Características:

- Fácil de instalar y configurar en la mayoría de plataformas.
- Independiente del sistema gestor de Base de Datos.
- Utiliza programación orientada a objetos, de ahí que sea imprescindible PHP 5.
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.

- Aunque utiliza MVC (Modelo vista controlador), tiene su propia forma de trabajo en este punto, con variantes del MVC clásico como la capa de abstracción de base de datos, el controlador frontal y las acciones.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Fácil de extender, lo que permite su integración con las bibliotecas de otros fabricantes.
- Es una estructura de librerías y clases para programar aplicaciones web. (10)

1.9.2 Kumbia.

KumbiaPHP es un framework para aplicaciones web libre escrito en PHP5. Basado en las prácticas de desarrollo web. Kumbia fomenta la velocidad y eficiencia en la creación y mantenimiento de aplicaciones web. Intenta proporcionar facilidades para construir aplicaciones robustas para entornos comerciales. Además es un esfuerzo por producir un framework que ayude a reducir el tiempo de desarrollo de una aplicación web sin producir efectos sobre los programadores.

Características:

- Mapeo Objeto Relacional (ORM) y Separación MVC.
- Soporte para AJAX.
- Generación de Formularios.
- Componentes Gráficos.
- URL amigables

1.9.3 Framework Seleccionado.

Los frameworks analizados anteriormente son para el desarrollo de aplicaciones web, a pesar de esto se decidió utilizar Symphony ya que está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symphony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows.

1.10 Gestor de Base de Datos.

Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos. Ayuda a realizar las siguientes acciones:

- Definición de los datos.
- Mantenimiento de la integridad de los datos dentro de la base de datos.
- Control de la seguridad y privacidad de los datos.
- Manipulación de los datos.

1.10.1 PostGreeSQL.

Sistema Gestor de bases de datos que almacena los datos de la aplicación es PostgreSQL. Es un sistema de base de datos relacional perteneciente al ámbito del software libre que se destaca por su robustez, escalabilidad y cumplimiento de los estándares SQL. Cuenta con versiones para una amplia gama de sistemas operativos, entre ellos: Linux, Windows, Mac SX, Solaris y otros más.

Características de PostGreeSQL.

- Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes, cadenas de bits, etc. También permite la creación de tipos propios.
- Incorpora una estructura de datos array.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

1.10.2 MySQL.

MySQL es un sistema de gestión de base de datos relacional. Opera en una arquitectura cliente/servidor, de tal manera que el servidor solo tiene que enviarle una cadena de caracteres y esperar la devolución de los datos. MySQL ha pasado de ser una pequeña base de datos a una completa herramienta. Es una tecnología útil para la creación de bases de datos seguras al poseer un sistema de privilegios y

contraseñas que es muy flexible y seguro, que permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de ellas a través de la Web está encriptado al conectarse con un servidor.

Las principales características de este gestor de bases de datos son las siguientes:

- Soporta gran cantidad de tipos de datos para las columnas.
- Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP).
- Relativamente sencillo de añadir otro sistema de almacenamiento. Es útil si desea añadir una interfaz SQL para una base de datos propia.
- Gran portabilidad entre sistemas.

Desventajas de MySQL:

- Carece de soporte para transacciones y subconsultas.
- El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación, sobre todo para aquellos programadores que provienen de otros gestores que sí poseen esta característica.
- No es viable para su uso con grandes bases de datos, a las que se acceda continuamente, ya que no implementa una buena escalabilidad.

1.10.3 Gestor de Base de Datos Seleccionado.

Se decidió utilizar como Gestor de Base de Datos PostGreeSQL ya que posee una gran escalabilidad. Es capaz de ajustarse al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta (en algunos benchmarks se dice que ha llegado a soportar el triple de carga de lo que soporta MySQL). Además implementa el uso de subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en las que MySQL no podría.

1.11 Servidor Apache.

El servidor Apache es un software que está estructurado en módulos. La configuración de cada módulo se hace mediante la configuración de las directivas que están contenidas dentro del módulo. Los módulos del Apache se pueden clasificar en tres categorías:

- Módulos Base: Módulo con las funciones básicas del Apache.
- Módulos Multiproceso: Son los responsables de la unión con los puertos de la máquina, aceptando las peticiones y enviando a los hijos a atender a las peticiones.
- Módulos Adicionales: Cualquier otro módulo que le añada una funcionalidad al servidor.

Las funcionalidades más elementales se encuentran en el módulo base, siendo necesario un módulo multiproceso para manejar las peticiones. Se han diseñado varios módulos multiproceso para cada uno de los sistemas operativos sobre los que se ejecuta el Apache, optimizando el rendimiento y rapidez del código.

El resto de funcionalidades del servidor se consiguen por medio de módulos adicionales que se pueden cargar. Para añadir un conjunto de utilidades al servidor, simplemente hay que añadirle un módulo, de forma que no es necesario volver a instalar el software.

Características:

- Funciona sobre muchas plataformas.
- Módulos cargados dinámicamente.
- Php3 + Bases de datos.
- SSL: transacciones seguras.
- Alto desempeño.

1.12 Patrones.

Pareja de problema / solución con un nombre, que codifica (estandariza) buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades.

Según Christopher Alexander[1] “Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma”. Dentro de la rama informática los patrones se clasifican en:

- Patrones de Caso de Uso.

[1] Christopher Alexander: Destacado Arquitecto reconocido internacionalmente por sus numerosos aportes en la Teoría de Patrones.

- Patrones Arquitectónicos.

1.12.1 Patrones de Caso de Uso.

La experiencia en la utilización de casos de uso ha evolucionado en un conjunto de patrones que permiten con más precisión reflejar los requisitos reales, haciendo más fácil el trabajo con los sistemas, y mucho más simple su mantenimiento. Dado un contexto y un problema a resolver, estas técnicas han mostrado ser la solución adoptada en la comunidad del desarrollo de software. Se presentan a modo de herramientas que permiten resolver los problemas que se les planteen a los desarrolladores de una forma ágil y sistemática. Estos patrones se enfocan hacia el diseño y las técnicas utilizadas en modelos de alta calidad, y no en como modelar casos específicos. Utilizando estos patrones, arquitectos, analistas, ingenieros, y gerentes pueden lograr mejores resultados de forma más rápida. (11)

Los patrones de caso de uso que se utilizarán son:

- Patrón CRUD (Completo).
- Patrón Múltiples Actores (Rol Común).

Patrón CRUD.

Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual, consta de un caso de uso, llamado Información CRUD o Gestionar información, modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y a su vez son cortos y simples.

Patrón Múltiples Actores.

Roles Comunes.

Puede suceder que los dos actores jueguen el mismo rol sobre el caso de uso. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del caso de uso, solo exista una entidad externa interactuando con cada una de las instancias del caso de uso.

PATRÓN MÚLTIPLES ACTORES, ROLES COMUNES.

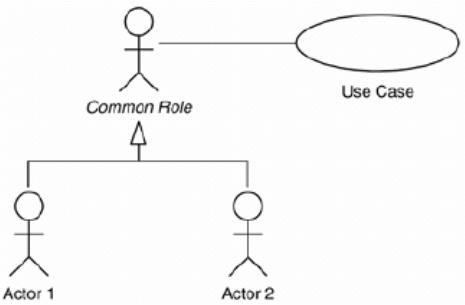

Figura 1.1 Patrón múltiples actores, Roles comunes.

1.12.2 Patrones de Diseño y Patrones de Arquitectura.

Dentro del marco de la arquitectura de software se insertan lógicamente los elementos relacionados al diseño del sistema en cuestión, en este caso, una aplicación para gestión de no conformidades y recursos, y análogamente a los patrones de arquitectura, también existen los patrones de diseño.

Patrones GoF

Se utilizará el framework Symphony para el desarrollo del sistema informático, este framework utiliza una serie de patrones GOF como son:

En la categoría Creacionales:

Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. En el controlador frontal hay una llamada a sfContext: getInstance (). En una acción, el método getContext (), un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symphony.

En la categoría Estructurales:

Decorator (Envoltorio): Añade funcionalidad a una clase, dinámicamente. El archivo layout.php, que también se denomina plantilla global, almacena el código HTML que es común a todas las páginas de la aplicación. (12)

Patrones GRASP

Creador.

En la clase Actions se encuentran las acciones definidas para el sistema y se ejecutan cada una de ellas. En las acciones se crean los objetos de las clases que representan las entidades, evidenciando de este modo que la clase Actions es "creador" de dichas entidades.

Experto.

Es uno de los más utilizados, puesto que Propel es la librería externa que utiliza Symphony para realizar su capa de abstracción en el modelo, encapsula toda la lógica de los datos y son generadas las clases con todas las funcionalidades comunes de las entidades.

Alta Cohesión.

Symphony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase Actions tiene la responsabilidad de definir las acciones para las plantillas y colabora con otras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades, es decir, está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el software sea flexible frente a grandes cambios.

Controlador.

Todas las peticiones web son manejadas por un solo controlador frontal (sf Actions), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL entrada por el usuario.

Bajo Acoplamiento.

La clase Action hereda solamente de sf Actions para lograr un bajo acoplamiento de clases.

Patrones de Arquitectura

En la última década cambió la visión que los desarrolladores tienen de los sistemas de software. Esta nueva visión se llamó "arquitectura". La arquitectura de software básicamente indica la estructura, funcionamiento e interacción entre las partes del software (13). Desde los pequeños programas hasta los

sistemas más grandes poseen una estructura y un comportamiento que los hace clasificables según su arquitectura.

Dentro de los patrones de arquitectura se encuentran el patrón Modelo Vista Controlador (MVC) y el patrón modelo de tres capas. Para el desarrollo de este trabajo de diploma como fue anteriormente especificado se hará uso del framework Symphony, el cual está basado en el patrón Modelo Vista Controlador.

MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. Divide una aplicación interactiva en 3 áreas: procesamiento, salida y entrada. Utiliza las siguientes abstracciones:

- Modelo: Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.
- Vista: Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- Controlador: Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón y pulsaciones de teclas. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista.

1.13 Conclusiones del Capítulo.

En este capítulo se abordó lo referente a la calidad de software, así como diferentes conceptos relacionados con las no conformidades, se analizaron las tendencias y tecnologías actuales ofreciendo las definiciones necesarias para comprender las herramientas que serán utilizadas en el desarrollo de la aplicación. Con el objetivo de obtener un proceso disciplinado sobre el desarrollo de software y con el fin de hacerlo más predecible y eficiente se empleará como metodología de desarrollo, RUP, la cual utiliza como lenguaje de modelado UML. Para obtener un mejor diseño se hará uso de la herramienta de modelado Visual Paradigm. Además para lograr una mayor facilidad a la hora de desarrollar la herramienta se utilizará PHP como lenguaje de programación donde el entorno de desarrollo a utilizar será NetBeans 6.8 Beta, el cual será integrado al framework de desarrollo Symphony y PostGreeSQL como gestor de base de datos.

Capítulo 2: Características del Sistema.

En este capítulo se refleja un análisis detallado de los procesos que serán objeto de automatización y los trabajadores que desarrollan dichos procesos. Se define la información que se maneja mediante un modelo de negocio y la propuesta del sistema. Además de identificar los requisitos funcionales y no funcionales, los casos de uso con sus descripciones, los diagramas de actividades y el modelo de objeto.

2.1. Objeto de Estudio.

2.1.1. Situación Problemática.

El grupo de calidad de la facultad 6 se ha trazado varios objetivos con el fin de controlar que el software elaborado presente el mínimo de errores posibles, así como formar estudiantes con el perfil de calidad de software. La actividad de la gestión de no conformidades comienza cuando el asesor de calidad asigna el trabajo al grupo de probadores que van a realizar las pruebas a determinado software. El probador comienza a revisar y a medida que detecta errores en los diferentes artefactos va insertando en un documento estas No Conformidades. Posteriormente el revisor líder envía estas No Conformidades al grupo de desarrollo, ellos le dan respuesta y luego son enviadas al equipo de prueba. Este proceso de gestión de las No Conformidades se hace de forma manual, no hay un control de la documentación ni de la seguridad de las mismas. Además no se encuentra centralizado ni organizado un control sobre los recursos tanto humanos como tecnológicos del grupo de calidad de la facultad .Todo esto conlleva a un mal uso de la documentación y pérdida de la información por lo que no se hace una buena gestión de las No Conformidades y Recursos.

2.1.2. Objeto de Automatización.

Grupo de Calidad.

- Control de la gestión de las No Conformidades encontradas al software.
- Control de los reportes de las No Conformidades.
- Control de los reportes de la cantidad de No Conformidades encontradas.
- Control de la gestión de los recursos.
- Control de la información.

2.1.3. Propuesta del sistema.

La herramienta que se propone desarrollar **SIGNCR** automatizará todos los procesos que se llevan a cabo en el grupo de calidad de la facultad. La herramienta gestionará todas las No Conformidades encontradas durante el proceso de revisión, para realizar esta actividad se debe tener previamente toda la documentación necesaria en dependencia del tipo de prueba que se realizará y las condiciones requeridas para el proceso de prueba. Las No Conformidades serán insertadas en el sistema y éstas se podrán modificar, eliminar. El sistema permitirá exportar los diferentes reportes de No Conformidades, asignar el trabajo a los probadores, los módulos y los proyectos y a la vez permite dar un reporte de toda la información relacionada con este proceso. Permite administrar todos los recursos utilizados para la realización de las diferentes pruebas, conociendo el estado real de ellos y a la vez brindar un conjunto de reportes que posibilitan mantener actualizados a los directivos de la facultad.

Esta herramienta se construirá sobre la tecnología PHP, utilizando las amplias ventajas que el mismo ofrece y que fueron descritas en el capítulo 1, además utilizará como gestor de base de datos PostGreeSQL por ser compatible con el lenguaje seleccionado y brindar diferentes funcionalidades descritas también en el capítulo 1. El sistema constará de políticas de seguridad otorgando a cada usuario los derechos que le corresponden. Existirán 5 tipos de usuarios: administrador, asesor de calidad, revisor líder, jefe de proyecto y gestor de reportes. Cada usuario que entra al sistema debe autenticarse antes de realizar alguna acción.

2.2. Reglas del Negocio.

Las reglas de negocio describen políticas que deben cumplirse o condiciones que deben satisfacerse, por lo que regulan algún aspecto del negocio. El proceso de especificación implica que hay que identificarlas dentro del negocio, evaluar si son relevantes dentro del campo de acción que se está modelando e implementarlas en la propuesta de solución.

Las reglas del negocio identificadas son:

- El sistema automatizará las siguientes funcionalidades.
    1. Control de la gestión de las No Conformidades encontradas al software.
    2. Control de los reportes de las No Conformidades.
    3. Control de los reportes de la cantidad de No Conformidades encontradas.

4. Control de la gestión de los recursos.
5. Control de la información.

- El sistema solo será utilizado por el laboratorio de calidad de la facultad.
- Solo pueden acceder al sistema el Asesor de Calidad, el Revisor Líder, el Gestor de Reportes, el administrador y el jefe de proyecto.
- En caso que un proyecto entre por segunda vez en fase de prueba se debe especificar la versión.

2.3. Modelo de Negocio.

El modelo de negocio que se describe responde a un conjunto de actividades realizadas dentro de los procesos que se desarrollan en el grupo de calidad de la facultad 6. Inicialmente el solicitante realiza la solicitud de revisión de sus productos al asesor de calidad. En caso de existir disponibilidad de revisión y que la solicitud cumpla con los requisitos establecidos se procede a la revisión del software, en caso contrario no se realiza este proceso, este caso de uso se puede insertar dentro del caso de uso Base (Realizar Solicitud). El caso de uso Revisar Software incluye todo el proceso de ejecución de las pruebas, gestión de las No Conformidades y seguimiento de los recursos involucrados. Los directivos (decano, jefe de centro, directivos de calidad, especialistas de Calisoft) pueden solicitar reportes en dependencia de su interés.

2.3.1. Actores del negocio.

Un actor del negocio es cualquier individuo, grupo, entidad, organización, máquina o sistema de información externos con los que el negocio interactúa. Lo que se modela como actor es el rol que se juega cuando se interactúa con el negocio para beneficiarse de sus resultados.

Actor	Descripción
Solicitante	Jefe de proyecto que realiza la solicitud para la revisión de sus productos.
Directivo	Decano, jefe de centro, directivos de calidad, especialistas de Calisoft que pueden solicitar reportes en dependencia de su interés.

Tabla 2.1 Descripción de los actores del negocio.

2.3.2. Trabajadores del Negocio.

Un trabajador del negocio es una abstracción de una persona (o grupo de personas), una máquina o un sistema automatizado que actúa en el negocio realizando una o varias actividades, interactuando con otros trabajadores del negocio y manipulando entidades del negocio. Representa un rol.

Trabajador	Descripción
Asesor de Calidad	Encargado de verificar el expediente del proyecto a probar, así como gestionar los recursos de prueba.
Probador	Encargados de llevar a cabo las pruebas y redactar las no conformidades.
Revisor Líder	Encargado de gestionar las no conformidades.

Tabla 2.2 Descripción de los trabajadores del negocio.

2.3.3. Diagrama de Caso de Uso del Negocio.

Figura 2.1 Diagrama del Modelo de Negocio.

2.3.4. Casos de Uso del Negocio.

- Realizar Solicitud.
- Revisar Software.
- Solicitar Información.

Realizar Solicitud: El caso de uso comienza cuando el solicitante realiza la solicitud de revisión de sus productos al asesor de calidad, se verifica la completitud de la solicitud, en caso de existir disponibilidad

de revisión indica revisar el software, en caso contrario le comunica al jefe de proyecto que el software no puede ser revisado en esos momentos.

Revisar Software: El caso de uso se inicia cuando el asesor de calidad indica revisar el software, se realiza el proceso de prueba y finaliza cuando se decide liberar o abortar la revisión.

Solicitar Información: El caso de uso comienza cuando el directivo decide solicitar un reporte, el asesor de calidad recoge toda la información necesaria y finaliza el caso de uso cuando el asesor de calidad emite el informe sobre el reporte solicitado.

2.3.5. Diagrama de Actividades.

DIAGRAMA DE ACTIVIDADES: REALIZAR SOLICITUD.

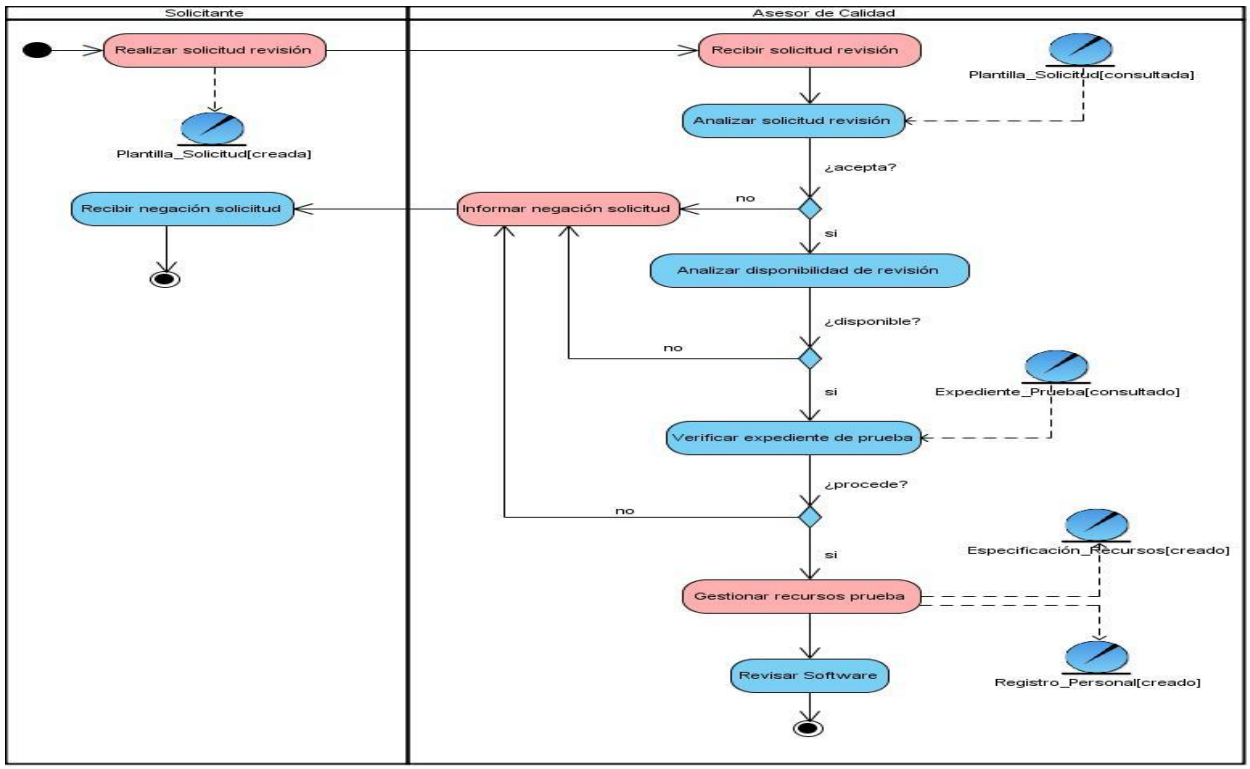

Figura 2.2 Diagrama de Actividades: Realizar Solicitud.

DIAGRAMA DE ACTIVIDADES: REVISAR SOFTWARE.

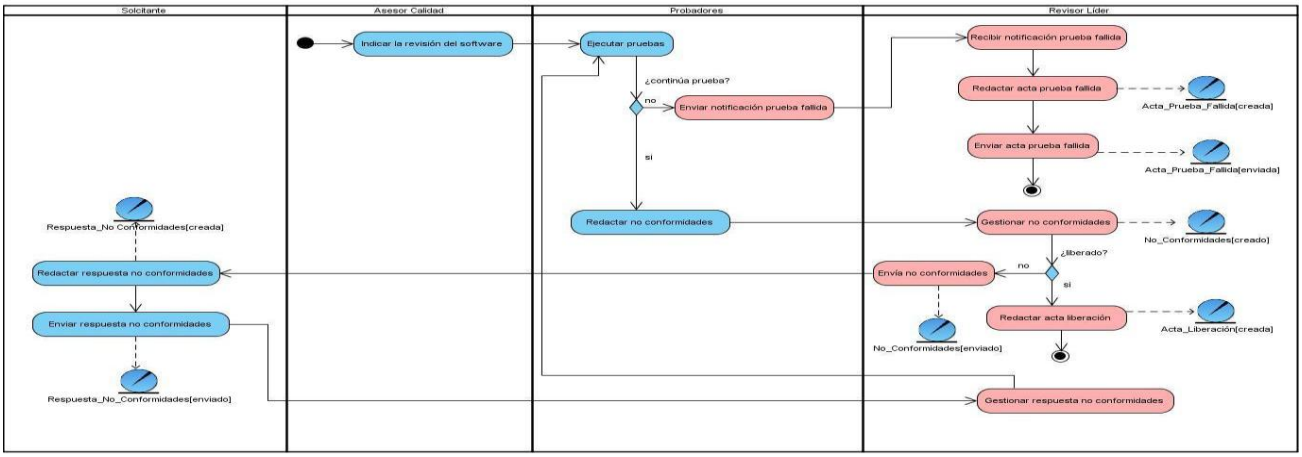

Figura 2.3 Diagrama de Actividades: Revisar Software.

DIAGRAMA DE ACTIVIDADES: SOLICITAR INFORMACIÓN

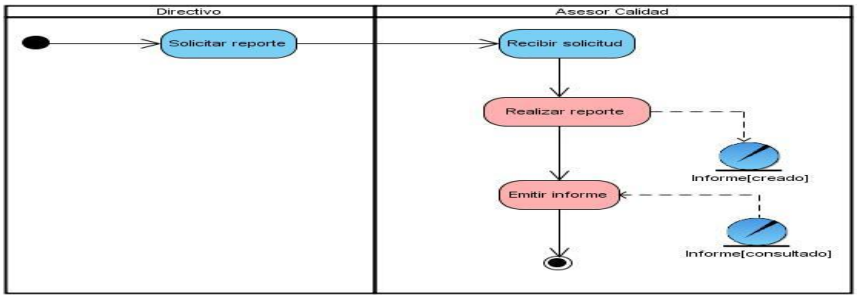

Figura 2.4 Diagrama de Actividades: Solicitar Información.

MODELO DE OBJETO: REALIZAR SOLICITUD.

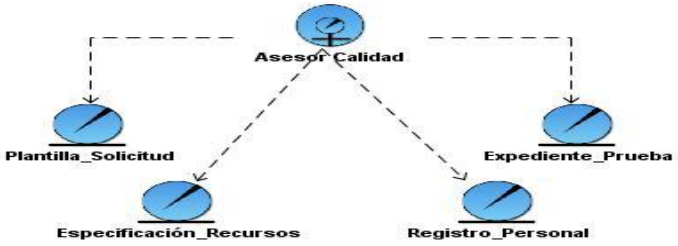

Figura 2.5 Modelo de Objeto: Realizar Solicitud.

MODELO DE OBJETO: REVISAR SOFTWARE.

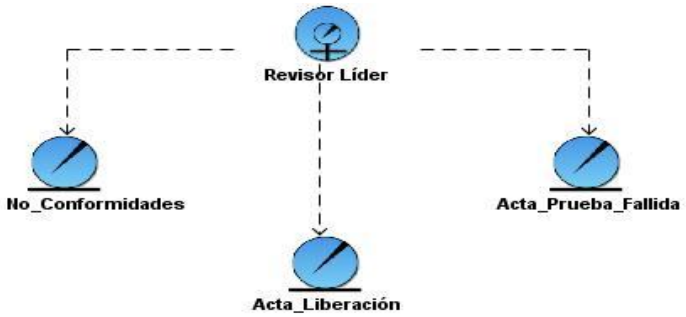

Figura 2.6 Modelo de Objeto: Revisar Software.

MODELO DE OBJETO: SOLICITAR INFORMACIÓN.

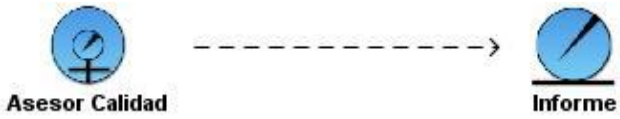

Figura 2.7 Modelo de Objeto: Solicitar Información.

2.4. Definición de Requisitos Funcionales.

Son capacidades o condiciones que el sistema debe cumplir. Se mantienen invariables sin importar con que propiedades o cualidades se relacionen.

RF 1: Autenticar Usuario.

RF 2: Gestionar Usuario.

- **RF 2.1:** Insertar nuevo usuario.
- **RF 2.2:** Buscar y visualizar usuario.
- **RF 2.3:** Modificar usuario.
- **RF 2.4:** Eliminar usuario.

RF 3: Realizar Solicitud de Prueba.

RF 4: Gestionar Proyecto.

- **RF 4.1:** Insertar Proyecto.
- **RF 4.2:** Buscar y visualizar Proyecto.
- **RF 4.3:** Modificar Proyecto.

RF 4.4: Eliminar Proyecto.

RF 5: Gestionar Módulo.

RF 5.1: Insertar Módulo.

RF 5.2: Buscar y visualizar Módulo.

RF 5.3: Modificar Módulo.

RF 5.4: Eliminar Módulo.

RF 6: Gestionar Artefacto.

RF 6.1: Insertar Artefacto.

RF 6.2: Buscar y visualizar Artefacto.

RF 6.3: Modificar Artefacto.

RF 6.4: Eliminar Artefacto.

RF 7: Gestionar Tipo de Prueba.

RF 7.1: Insertar Tipo de Prueba.

RF 7.2: Buscar y visualizar Tipo de Prueba.

RF 7.3: Modificar Tipo de Prueba.

RF 7.4: Eliminar Tipo de Prueba.

RF 8: Gestionar Recursos Humanos.

RF 8.1: Insertar Recursos Humanos.

RF 8.2: Buscar y visualizar Recursos Humanos.

RF 8.3: Modificar Recursos Humanos.

RF 8.4: Eliminar Recursos Humanos.

RF 9: Gestionar Recursos Tecnológicos.

RF 9.1: Insertar Recursos Tecnológicos.

RF 9.2: Buscar y visualizar Recursos Tecnológicos.

RF 9.3: Modificar Recursos Tecnológicos.

RF 9.4: Eliminar Recursos Tecnológicos.

RF 10: Gestionar No Conformidades.

RF 10.1: Insertar No Conformidades.

RF 10.2: Buscar y visualizar No Conformidades.

RF 10.3: Modificar No Conformidades.

RF 10.4: Eliminar No Conformidades.

RF 11: Generar Reportes.

RF 12: Generar Acta de Liberación.

RF 13: Generar Acta de Prueba Fallida.

2.5. Definición de Requisitos No Funcionales.

Un producto software puede cumplir con todas las funcionalidades requeridas, pero si no es un software seguro y no cumple con las propiedades no funcionales, se puede asegurar que no es un software confiable y por tanto los clientes no quedan satisfechos con los resultados, para resolver estas incongruencias se establecen los requerimientos no funcionales, siendo propiedades o cualidades que el producto software debe tener, ejemplo de ello son las restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, entre otras.

1. **Apariencia o Interfaz Externa.**
   - El sistema deberá poseer una interfaz web sencilla, amigable, lo más atractiva y clara posible para el usuario, además su funcionamiento debe ser de fácil comprensión.

2. **Usabilidad.**
   - La aplicación garantiza una fácil interacción entre cliente y PC, de tal forma que no haya conflictos de usabilidad entre ambos.
   - El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora y de un ambiente Web en sentido general.

3. **Rendimiento.**
   - Debido a que se trata de una aplicación cliente/servidor debe ser eficiente, con capacidad adecuada de procesamiento y cálculo, así como requiere de un tiempo de respuesta relativamente pequeño.

4. **Portabilidad.**
   - El sistema debe ser multiplataforma.

5. **Seguridad.**
    - Para la entrada de usuarios al sistema debe ser verificado si el mismo ya está autenticado, si no lo está brindarle el servicio de autenticación.

6. **Software.**
    - En el lado del Cliente debe existir un navegador que soporte JavaScript y Adobe Reader 5 o superior. En el lado del Servidor debe estar instalado el gestor de base de datos PostGreeSQL.

7. **Hardware.**
    - Procesador Pentium II o superior.
    - 256 MB de memoria RAM o superior.
    - Impresora.

2.6. Actores del Sistema.

Cada trabajador del negocio (incluso si fuera un sistema ya existente) que tiene actividades a automatizar es un candidato a actor del sistema. Si algún actor del negocio va a interactuar con el sistema, entonces también será un actor del sistema.

Actor	Descripción
Administrador	Encargado del mantenimiento del sistema, así como de gestionar todo el proceso de permisos a los usuarios que acceden al mismo. Este rol constituye una generalización del Usuario del sistema.
Usuario	Rol que representa a los usuarios del sistema que se han autenticado y pueden acceder a los recursos que le son permitidos.
Jefe de Proyecto	Este rol constituye una generalización del Gestor de Reportes. Además es el encargado de solicitar la revisión de sus productos.
Asesor de Calidad	Este rol constituye una generalización de Gestor de Reportes. Además es el encargado de gestionar los proyectos, artefactos y módulos que serán objeto de revisión, así como los recursos humanos y tecnológicos.
Revisor Líder	Este rol constituye una generalización del Gestor de Reportes. Es el responsable de gestionar las no

	conformidades, gestionar los tipos de pruebas, generar el acta de prueba fallida y acta de liberación.
Gestor de Reportes	Este rol constituye una generalización de Usuario, es el encargado de generar los reportes.

Tabla 2.3 Descripción de los actores del Sistema.

2.7. Listado de Casos de Uso.

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema.

CU-1	**Autenticar Usuario.**
Actor	Usuario.
Descripción	En este caso de uso los usuarios se pueden autenticar en el sistema.
Referencia	RF 1.

Tabla 2.4 Caso de Uso ¨Autenticar Usuario¨.

CU-2	**Gestionar Usuario.**
Actor	Administrador.
Descripción	En este caso de uso se insertan, modifican y eliminan los usuarios, así como asignarle los roles que existen en el sistema.
Referencia	RF 2.

Tabla 2.5 Caso de Uso ¨Gestionar Usuario¨.

CU-3	**Realizar Solicitud de Prueba.**
Actor	Jefe de Proyecto.
Descripción	En este caso de uso se realiza la solicitud de revisión de los productos.
Referencia	RF 3.

Tabla 2.6 Caso de Uso ¨Realizar Solicitud de Prueba¨.

CU-4	**Gestionar Proyecto.**
Actor	Asesor de Calidad.

Descripción	En este caso de uso se inserta, modifica, busca o elimina un proyecto.
Referencia	RF 4.

Tabla 2.7 Caso de Uso ¨Gestionar Proyecto¨.

CU-5	**Gestionar Módulo.**
Actor	Asesor de Calidad.
Descripción	En este caso de uso se inserta, modifica, busca o elimina un módulo.
Referencia	RF 5.

Tabla 2.8 Caso de Uso ¨Gestionar Módulo¨.

CU-6	**Gestionar Artefacto.**
Actor	Asesor de Calidad.
Descripción	En este caso de uso se inserta, modifica, busca o elimina un artefacto.
Referencia	RF6.

Tabla 2.9 Caso de Uso ¨Gestionar Artefacto¨.

CU-7	**Gestionar Tipo de Prueba.**
Actor	Revisor Líder.
Descripción	En este caso de uso se inserta, modifica, busca o elimina un tipo de prueba.
Referencia	RF 7.

Tabla 2.10 Caso de Uso ¨Gestionar Tipo de Prueba¨.

CU-8	**Gestionar Recursos Humanos.**
Actor	Asesor de Calidad.
Descripción	En este caso de uso se inserta, modifica, busca o elimina un recurso humano.
Referencia	RF 8.

Tabla 2.11 Caso de Uso ¨Gestionar Recursos Humanos¨.

CU-9	**Gestionar Recursos Tecnológicos.**
Actor	Asesor de Calidad.
Descripción	En este caso de uso se inserta, modifica, busca o elimina un recurso tecnológico.
Referencia	RF 9.

Tabla 2.12 Caso de Uso ¨Gestionar Recursos Tecnológicos¨.

CU-10	**Gestionar No Conformidades.**
Actor	Revisor Líder.
Descripción	En este caso de uso se insertan, modifican y eliminan los No Conformidades asociadas a un Proyecto, Módulo y Tipo de prueba.
Referencia	RF 10.

Tabla 2.13 Caso de Uso ¨Gestionar No Conformidades¨.

CU-11	**Generar Reportes.**
Actor	Gestor de Reportes.
Descripción	En este caso de uso se puede exportar un reporte.
Referencia	RF 11.

Tabla 2.14 Caso de Uso ¨Generar Reportes¨.

CU-12	**Generar Acta de Liberación.**
Actor	Revisor Líder.
Descripción	En este caso de uso se redacta el acta de liberación de un proyecto, además se puede exportar el acta de liberación.
Referencia	RF 12.

Tabla 2.15 Caso de Uso ¨Generar Acta de Liberación¨.

CU-13	**Generar Acta de Prueba Fallida.**
Actor	Revisor Líder.
Descripción	En este caso de uso se redacta el acta de prueba fallida de un proyecto, además se puede exportar el acta de prueba fallida.
Referencia	RF 13.

Tabla 2.16 Caso de Uso ¨Generar Acta de Prueba Fallida¨.

2.8. Diagrama de Caso de Uso del Sistema.

Un diagrama de casos de uso del sistema representa gráficamente a los procesos y su interacción con los actores.

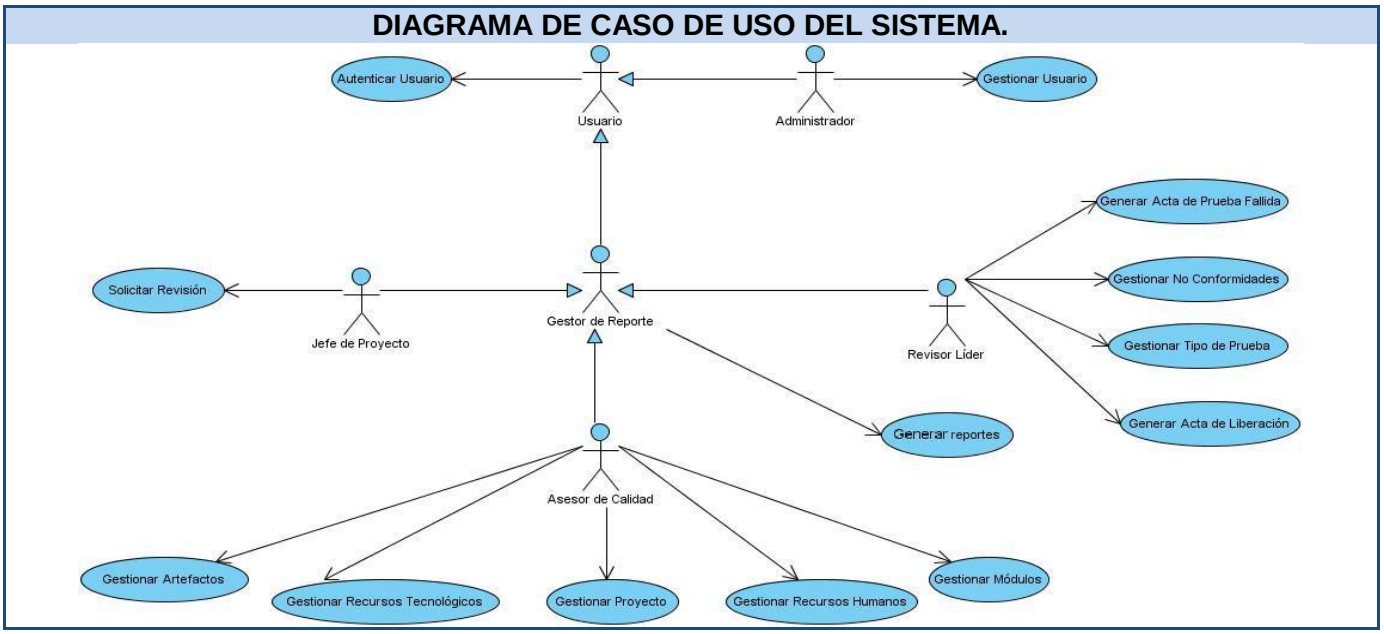

Figura 2.8 Diagrama de Caso de Uso del Sistema.

2.9. Conclusiones del Capítulo.

En este capítulo se definieron los principales procesos que serán objeto de automatización, los cuales se representaron a través de un modelo de negocio. Se identificaron 35 requisitos funcionales y no funcionales, así como los actores del sistema. Para un mejor conocimiento de las funcionalidades de la aplicación se realizaron las descripciones textuales de los 13 casos de uso, de ellos 4 son críticos.

Capítulo 3: Análisis y Diseño del Sistema.

En el presente capítulo se aborda otro flujo de trabajo de RUP, en este caso el Análisis y Diseño del sistema; a través de los artefactos que se generan en este flujo se modelan los casos de usos seleccionados. Se realizan las clases del análisis, así como los diagramas de colaboración para los casos de uso arquitectónicamente significativos. Además se analizan los casos de uso del sistema para diseñar las clases que se implementarán, se representan los diagramas de secuencia del diseño, el diagrama de las clases diseñadas con sus relaciones, los principios utilizados para el diseño de dichas clases, el diagrama de clases persistentes, el diagrama entidad relación y la descripción de las tablas de la base de datos.

3.1. Análisis.

Durante el flujo de trabajo de análisis se hace un estudio minucioso de los requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una compresión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema entero, incluyendo su arquitectura, es importante hacer notar que el análisis hace abstracciones y evita resolver algunos problemas y tratar algunos requisitos que serán pospuestos al diseño y la implementación.

3.1.1. Modelo de Análisis.

El Modelo de análisis es una jerarquía de paquetes del análisis que contienen clases del análisis y realizaciones de casos de uso, además ofrece una especificación más precisa de los requisitos, sin dejar de mencionar que en el modelo de análisis se describe utilizando el lenguaje de los desarrolladores, puede por tanto introducir un mayor formalismo y ser utilizado para razonar sobre los funcionamientos internos del sistema. El modelo de análisis estructura los requisitos de un modo que facilita su comprensión, su preparación, su modificación y en general su mantenimiento; además se puede considerar como la primera aproximación al modelo de diseño. (14)

3.1.2. Diagrama de Clases del Análisis.

Un diagrama de clases del análisis es un artefacto en el que se representan los conceptos en un dominio del problema.

DIAGRAMA DE CLASES DEL ANÁLISIS: REALIZAR SOLICITUD.

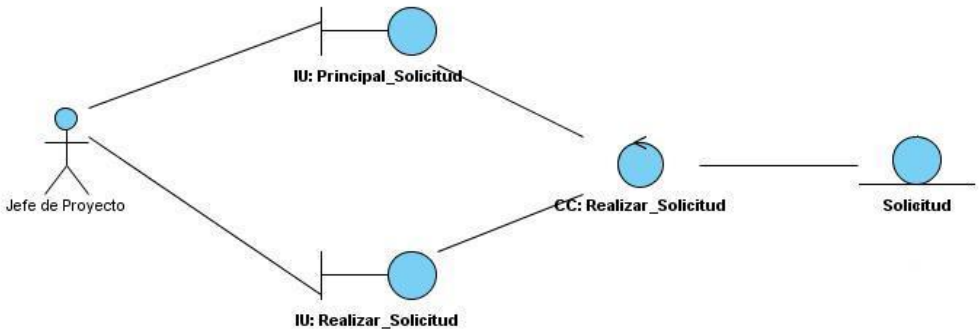

Figura 3.1 Diagrama de Clases del Análisis del Caso de Uso Realizar Solicitud.

3.1.3. Diagramas de Colaboración del Análisis.

DIAGRAMA DE COLABORACIÓN DEL ANÁLISIS: REALIZAR SOLICITUD.

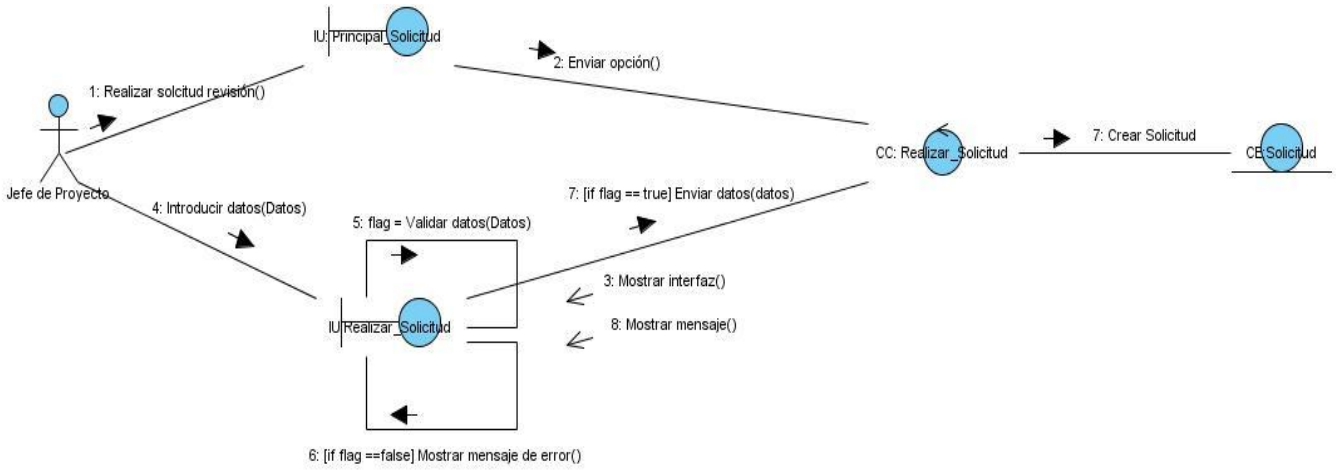

Figura 3.2 Diagrama de Colaboración del Análisis del Caso de Uso Realizar Solicitud.

3.2. Descripción de estilos arquitectónicos y patrones de diseño.

Los patrones arquitectónicos especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes.

Symphony como framework de desarrollo que se emplea toma lo mejor de la arquitectura MVC e implementa de forma que el desarrollo de aplicaciones sea rápido y sencillo representándolo a través de tres elementos fundamentales: el modelo, la vista y el controlador.

La **capa del modelo** define la lógica de negocio, la base de datos pertenece a esta capa. Encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida y/o comportamiento de entrada. Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura su integridad y permite derivar nuevos datos.

La **vista** es lo que utilizan los usuarios para interactuar con la aplicación, los gestores de plantillas pertenecen a esta capa. En Symphony la capa de la vista está formada principalmente por plantillas en PHP. Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

El **controlador** recibe las entradas, traducidas a solicitudes de servicio para el modelo. Es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

3.2.1. Patrones de diseño.

Al usar un framework de PHP para desarrollar, se tiene que adoptar la metodología que impone, una de las principales ventajas a señalar en el uso de los framework es que están basados en patrones de diseño, siendo una de las características que hace posible la gran usabilidad que tienen los framework, casi siempre independiente del tipo de aplicación Web que se desea desarrollar. Symphony no queda exento de esto, y está concebido de tal manera que obliga al programador aplicar diferentes patrones de diseño.

Son muchos los patrones que se utilizan en la implementación con Symphony, a continuación se mencionan algunos ejemplos de los evidenciados, ubicándolos en las capas de Modelo y Control que plantea el patrón arquitectónico MVC.

Patrón Experto

En la arquitectura de Symphony, específicamente en el modelo, existen dos tipos de clases fundamentales:

- Las clases encargadas de la abstracción de datos (responsables de realizar todas las operaciones con la BD).
- Las de acceso a datos (responsables de interactuar con las clases de abstracción de datos, devuelven los objetos que necesitan los controladores en su forma original).

Symphony genera 4 clases por cada tabla de la BD, por ejemplo en la presente aplicación se tiene una tabla denominada NoConformidades, se generan las siguientes clases: NoConformidades, Base NoConformidades, NoConformidadesPeer y BaseNoConformidadesPeer. De estas cuatro clases, se percibe que las clases que trabajan directamente con la BD, son las terminadas en Peer, estas clases son las encargadas de hacer las consultas a la BD utilizando Propel, por tanto estas clases (en nuestro ejemplo, BaseNoConformidades y BaseNoConformidadesPeer) son las clases de abstracción de datos. Como clases de abstracción de datos son las que tienen entonces los atributos necesarios para realizar dicha función, por tanto deben implementar la responsabilidad de realizar las acciones directamente con la Base de Datos y aquí es donde se aplica el patrón Experto.

Según los atributos de la clase BaseNoConformidadesPeer están más relacionado con la Base de Datos (nombre de la tabla, cantidad de columnas, etc.) y BaseNoConformidades más bien se basa en tener los datos meramente de la clase que responde a esa tabla y no a la tabla específicamente, en el fragmento del método delete de la clase BaseNoConformidades que invoca al método doDelete de la Peer, aquí es donde se ve el Experto, ella no conoce los atributos para comenzar a interactuar con la Base de Datos tanto sólo le “avisa” a la Peer lo que tiene que hacer, la Peer si tiene todos los datos necesarios para ejecutar esta acción.

Patrón Creador.
En Symphony hay clases que aplican el patrón Singleton, y por tanto la creación de los objetos de estas clases se hacen una sola vez y los métodos de estas clases son estáticos, un ejemplo de esto es en las clases “action” cuando desea crear una instancia de una clase del Modelo, por ejemplo se tiene en la clase “action” del módulo no_conformidades, la acción Visualizar Datos toma el Id de una no conformidad y desea ver los datos de la misma, se necesita crear una instancia de esa no conformidad. En este caso el “action” usa al objeto NoConformidadesPeer, por tanto puede crear instancias de él evidenciándose este patrón.

Bajo Acoplamiento.

Este patrón se evidencia dentro del framework Symphony en la capa modelo ya que las clases de acceso a los datos tienen bastante independencia de las clases de abstracción de datos. Hay poca dependencia entre esas clases lo que permite una mayor reutilización.

Alta Cohesión.

Una de las características principales del framework Symphony es la organización del trabajo en el mismo en cuanto a la estructura del proyecto, lo cual permite crear y trabajar con clases con una alta cohesión. Por ejemplo, la clase **Actions** del módulo **no_conformidades** contiene varias funcionalidades en las que cada una posee un propósito único, no desempeñado por el resto de los elementos, siendo estas funcionalidades las encargadas de controlar las acciones de las plantillas. Esto hace posible que el software sea flexible a cambios sustanciales con efecto mínimo.

Patrón Controlador.

Un ejemplo del patrón Controlador es fácil de encontrar, por ejemplo se puede ver desde la clase sfFrontController, sfFrontWebController, sfContex, los “actions”, el index.php del ambiente, etc. La arquitectura del framework (MVC) nos ayuda desde el principio, hay una capa específicamente para los controladores, que son el núcleo del mismo. Symphony aplica el patrón “Front Controller” (Controlador frontal) y por tanto tiene una estructura bien organizada de controladores, que parte desde el “index.php” del ambiente y terminan en los “actions”. Aquí cada clase en esta capa tiene su responsabilidad y es única, hay controladores que se encargan de la seguridad del sistema trabajando con ficheros YML, otros solo velan por identificar mediante unos datos las clases que se deben realizar determinadas tareas.

Patrones GoF.

Patrón Decorador.

El uso de este patrón se pone de manifiesto en la relación que se establece entre el layout o plantilla global y las diferentes plantillas que forman parte de la vista. El archivo llamado layout.php que contiene el Layout de la página, almacena el código HTML que es común a todas las páginas de la aplicación, para no tener que repetirlo en cada página. El contenido de la plantilla se integra en el Layout, o si se mira desde otro punto de vista, el Layout decora la plantilla. Este comportamiento es una implementación del patrón de diseño llamado “Decorator”.

Singleton (Instancia única).

Este patrón se aplica en el método getInstance de la clase sfRouting, el cual garantiza que esa clase solo tenga una única instancia, proporcionando un punto de acceso global a la misma.

La clase sfRouting es una de las que utiliza el controlador frontal (sfFrontWebController), es muy utilizada porque es la encargada de enrutar todas las peticiones que se hagan a la aplicación. La clase sfRouting define otros métodos muy útiles para la gestión manual de las rutas: ClearRoutes (), hasRoutes (), getRoutesByName

3.3. Diagrama de clases del diseño.

Los diagramas de clases son ampliamente utilizados en el modelado de sistemas orientados a objetos empleándose para representar las relaciones que se establecen entre las clases. Los diagramas de clases del diseño se utilizan para modelar la vista de diseño estática de un sistema, siendo la vista la representación a través de los diagramas de interacción.

DIAGRAMA DE CLASES DEL DISEÑO: REALIZAR SOLICITUD.

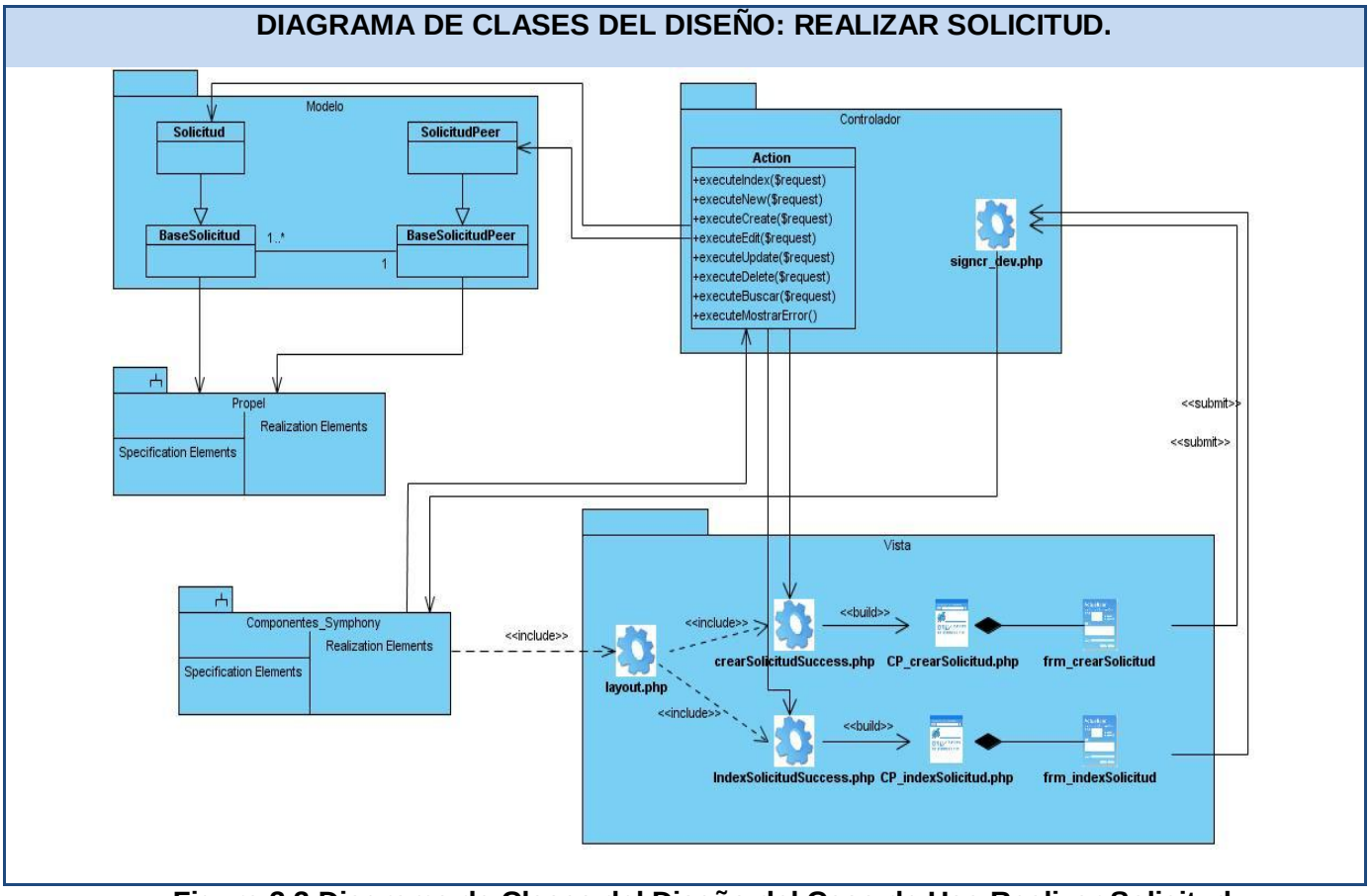

Figura 3.3 Diagrama de Clases del Diseño del Caso de Uso Realizar Solicitud.

3.4. Diagrama de Interacción del diseño. Secuencia.

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de los sistemas. Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes; un diagrama de colaboración es un diagrama de interacción que destaca la organización estructural de los objetos que envían y reciben mensajes. Para la realización de los casos de uso del diseño es más factible el empleo de los diagramas de secuencia ya que representan con más claridad el flujo de las acciones que debe realizar el sistema.

DIAGRAMA DE SECUENCIA DEL DISEÑO: REALIZAR SOLICITUD.

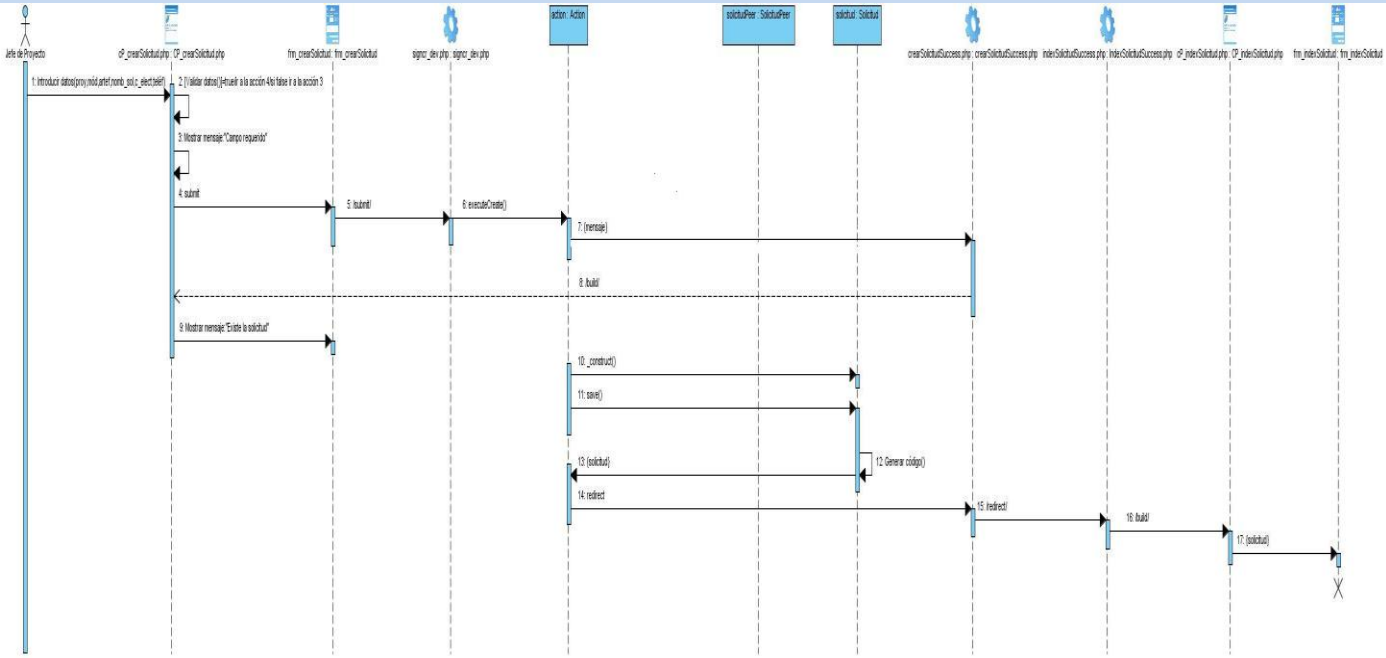

Figura 3.4 Diagrama de Secuencia del Diseño del Caso de Uso Realizar Solicitud.

3.5. Diagrama de Clases Persistentes.

Todas las clases identificadas en el dominio del análisis no son persistentes. La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Es responsabilidad del diseñador de definir cuales clases son las que deben ser persistentes. (15)

DIAGRAMA DE CLASES PERSISTENTES.

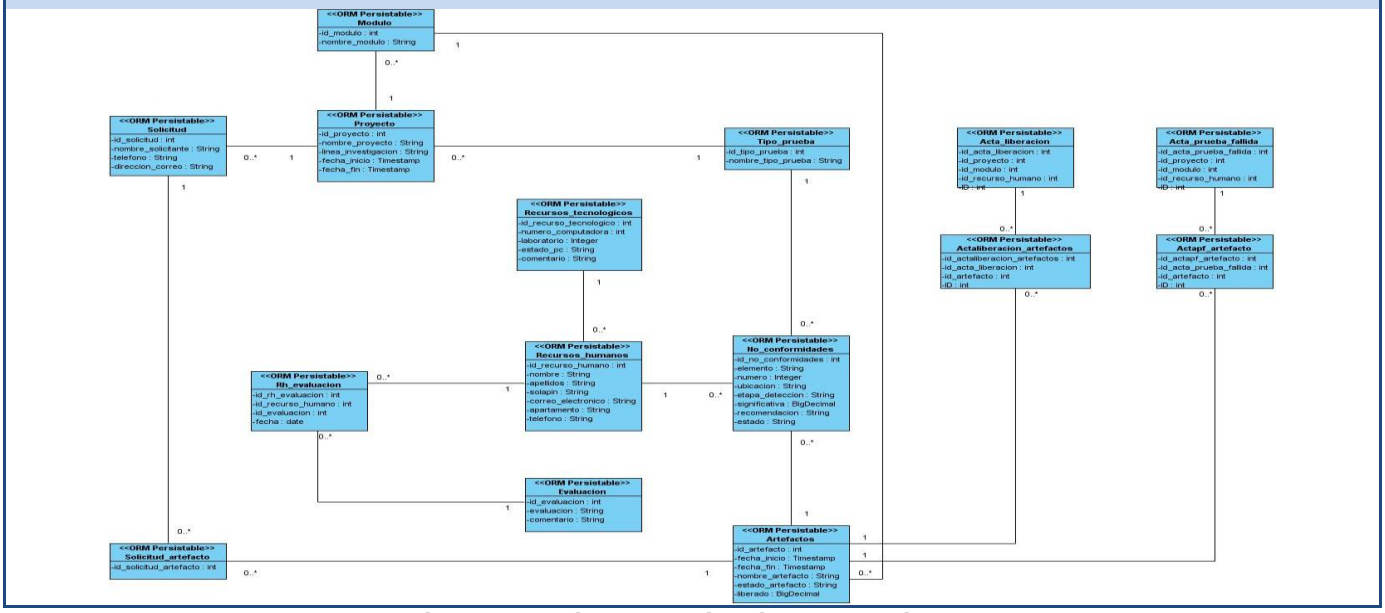

Figura 3.5 Diagrama de Clases Persistentes.

3.6. Modelo de Datos.

Colección de conceptos que sirven para describir la estructura de una base de datos, además de proporcionar los medios necesarios para conseguir la abstracción de los datos.

MODELO DE DATOS.

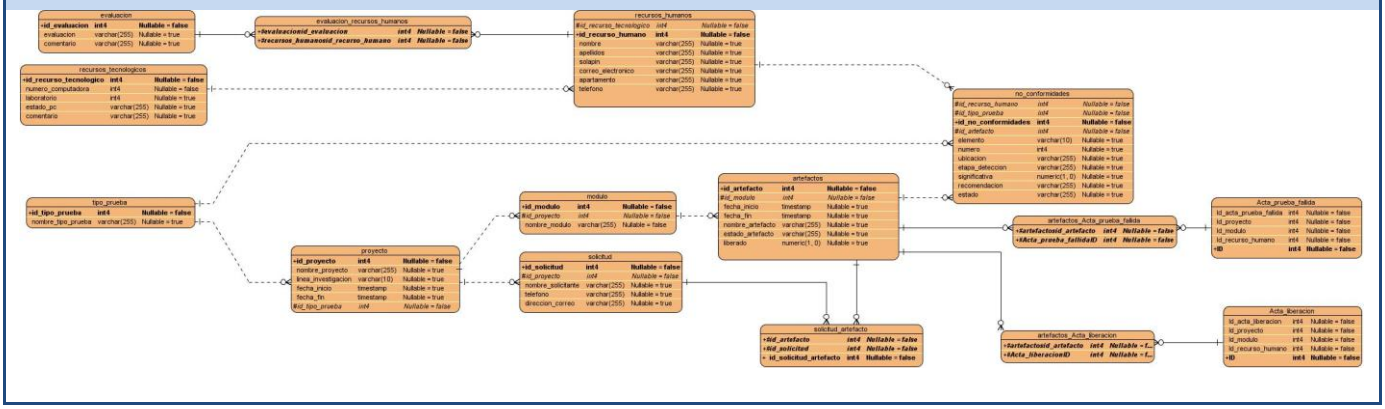

Figura 3.6 Modelo de Datos.

3.7. Modelo de Despliegue.

El Modelo de Despliegue se utiliza para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos.

El diagrama de despliegue realizado representa cuatro nodos principales. El nodo PC_Cliente que requiere de un navegador que soporte JavaScript y Adobe Reader 5.0 o superior, el nodo Servidor_Web, en el cual debe estar instalado el servidor Apache, en el nodo Servidor_BD debe estar instalado el Sistema Gestor de Base de Datos PostGreeSQL y el nodo dispositivo Impresora.

El nodo PC_Cliente estará conectado al nodo dispositivo Impresora mediante el protocolo de comunicación Universal Serial Bus USB y a su vez estará conectado mediante el Protocolo de Transferencia de Hipertexto HTTP al nodo procesador que representa al Servidor Web. La conexión entre el Servidor Web y el Servidor de Base de Datos se realizará mediante el protocolo de comunicación TCP/IP.

MODELO DE DESPLIEGUE.

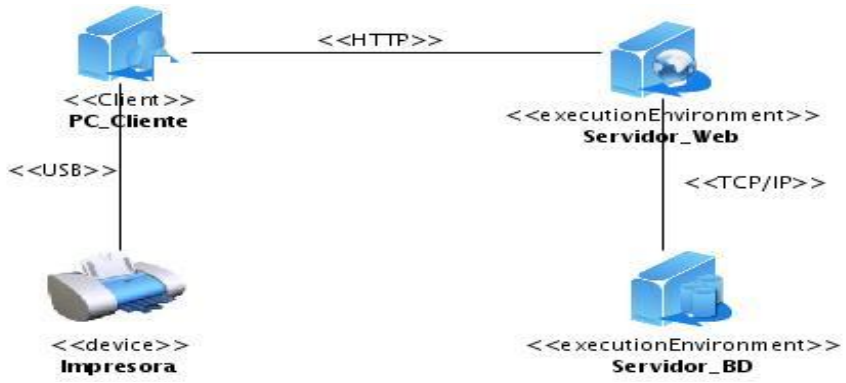

Figura 3.7 Modelo de Despliegue.

3.8. Conclusiones del Capítulo.

En este capítulo se hizo uso del patrón MVC, al igual que la aplicación de patrones de diseño, se definió la estructura del sistema mediante el flujo de trabajo Análisis y Diseño, en el análisis se realizaron los diagramas de clases y los diagramas de colaboración. En el diseño el diagrama de clases del diseño, diagrama de secuencia de los diferentes escenarios de cada caso de uso, el diagrama de clases persistentes, obteniéndose el modelo de datos, lo que facilitó el diseño de 14 tablas de la base de datos. Además se realizó el diagrama de despliegue que propició una visión de como está distribuido el sistema físicamente.

Capítulo 4: Implementación y Prueba

Durante este capítulo se realiza el flujo de trabajo de implementación, se analizan los principales artefactos como el Modelo de Implementación que incluye componentes, subsistemas de implementación y diagramas de componentes. Se comienza a implementar el sistema en términos de componentes mediante las clases del diseño. Además cuando se haya terminado el software se realizarán las pruebas.

4.1. Modelo de Implementación.

El Modelo de implementación representa la composición física de la implementación en términos de subsistemas de implementación y elementos de implementación. Describe como los elementos de diseño se implementan en componentes.

Se considera el artefacto más significativo del flujo de trabajo de Implementación, debido a la importancia que tiene para los desarrolladores comprender el funcionamiento del sistema desde el punto de vista de componentes y sus relaciones. Este modelo está conformado por el diagrama de componentes. (16)

4.1.1. Diagrama de Componentes.

Dentro del Modelo de Implementación se encuentran los diagramas de componentes. Un componente es la parte modular de un sistema, desplegable y reemplazable que encapsula implementación y un conjunto de interfaces y proporciona la realización de los mismos. El diagrama de componentes describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen los componentes unos de otros.

DIAGRAMA DE COMPONENTES: REALIZAR SOLICITUD.

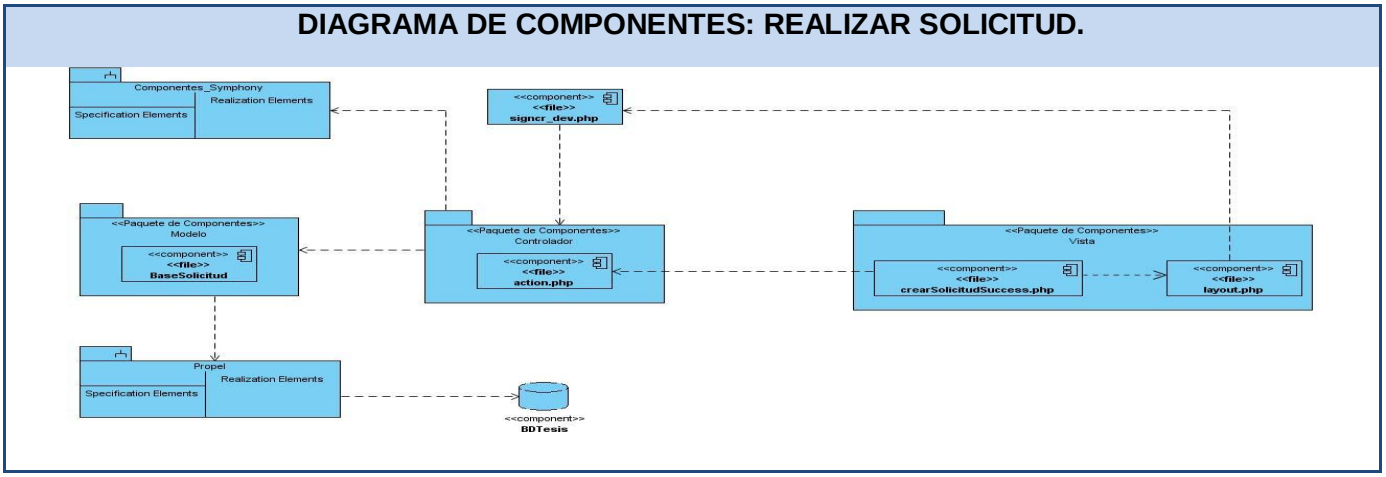

Figura 4.1 Diagrama de Componentes: Caso de Uso Realizar Solicitud.

4.2. Código Fuente.

Para obtener una versión funcional de la aplicación se deben implementar los componentes que se han definido, como resultado se obtienen archivos que contienen el código fuente de la aplicación. Es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

4.2.1. Estándares de Codificación.

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad, es de gran importancia para la calidad del software y para obtener un buen rendimiento. (17)

Estilo de Codificación Utilizado.

- Todas las etiquetas php deben ser completas (<?php?>)... no reducidas (<? ?>).
- Los bloques de código siempre deben estar encerrados por llaves (incluso si solo constan de una línea).
- Tamaño = 4 (espacios) para:
  - ✓ Declaraciones dentro de las clases.
  - ✓ Enunciado dentro de métodos y funciones.
  - ✓ Enunciados dentro de bloques de comandos.

4.2.2. Ejemplo de Código Fuente.

La mayoría de los métodos implementados responden a funciones básicas de inserción, actualización, eliminación, que se vuelven realmente sencillos mediante el uso que hace Symphony de Propel para realizar el mapeo objeto-relacional. Genera la mayor parte del código de acceso a datos proporcionando una abstracción al punto que permite que los implementadores tengan que utilizar muy pocas consultas a la base de datos.

A continuación se muestra el fragmento de código del método Buscar No Conformidades y se ofrece una breve descripción del mismo.

EJEMPLO DE CÓDIGO FUENTE. MÉTODO BUSCAR NO CONFORMIDADES

```
public function executeIndex(sfWebRequest $request)
{
	$this->numeroNC = trim($this->getRequestParameter('idnoconformidades'));
	$criterionc = new Criteria();
	if($this->numeroNC!='')
		$criterionc->add(NoConformidadesPeer::NUMERO,"$this->numeroNC%",Criteria::LIKE);

	$recursoH = RecursosHumanosPeer::doSelect(new Criteria());
	$this->recursosH = array();
	foreach($recursoH as $recursoh)
	{
		$this->recursosH[$recursoh->getIdRecursoHumano()] = $recursoh->getNombre();
	}

	$pruebas = TipoPruebaPeer::doSelect(new Criteria());
	$this->tipos = array();
	foreach($pruebas as $prueba)
	{
		$this->tipos[$prueba->getIdTipoPrueba()] = $prueba->getNombreTipoPrueba();
	}

	$artefactos = ArtefactosPeer::doSelect(new Criteria());
	$this->artefacto = array();
	foreach($artefactos as $artefact)
	{
		$this->artefacto[$artefact->getIdArtefacto()] = $artefact->getNombreArtefacto();
	}

		$this->no_conformidades_list = NoConformidadesPeer::doSelect($criterionc);
}
```

Figura 4.2 Ejemplo de Código Fuente: Método Buscar No Conformidades.

El método Buscar permite realizar una búsqueda de una no conformidad según el número de la misma, mostrando así como resultado la no conformidad que coincida con el número especificado.

4.3. Validación.

Symphony como framework empleado propone la validación de datos a través de la acción, que normalmente se corresponden con los parámetros de la petición. Symphony incluye un sistema de validación, utilizando métodos de la clase acción.

Cuando un usuario realiza una petición a una acción, Symphony siempre busca primero en la clase llamado **validateNombredelaAccion(),** si lo encuentra ejecuta esa clase. El valor de retorno de esta validación determina el siguiente método que se ejecuta: si devuelve true, entonces se ejecuta el método executeNombredelaAccion (); si el resultado es false entonces se ejecuta handleErrorNombredelaAccion (). En el caso de que handleNombredelaAccion () no exista, Symphony busca un método genérico llamado handleError (). Si tampoco existe, simplemente devuelve el valor sfView: ERROR para producir la plantilla NombredelaAccionError.php.

A continuación se muestra un ejemplo de como se ha validado un formulario de nuestro Sistema, perteneciente al Caso de Uso Gestionar Recursos Humanos.

FRAGMENTO DE CÓDIGO DE UNA VALIDACIÓN DE UN FORMULARIO

```
$this->setValidators(array(
  'id_recurso_humano'      => new sfValidatorPropelChoice(array('model' => 'RecursosHumanos',
  'column' => 'id_recurso_humano','required' => false)),
  'id_recurso_tecnologico' => new sfValidatorPropelChoice(array('model' => 'RecursosTecnologicos',
  'column' => 'id_recurso_tecnologico'),array('required'=>'Campo Requerido')),
  'nombre'                 => new sfValidatorString(array('max_length' => 255),
  array ('required' => 'Campo Requerido')),
  'apellidos'              => new sfValidatorString(array('max_length' => 255),
  array('required' => 'Campo Requerido')),
  'solapin'                => new sfValidatorString(array('max_length' => 255),
  array('required' => 'Campo Requerido')),
  'correo_electronico'     => new sfValidatorEmail(array('required'=>false),
  array('invalid' => 'La dirección de correo no es válida.')),
  'apartamento'            => new sfValidatorInteger(array('required' => false),
  array('invalid' => 'Datos Inválidos, solo números.')),
  'telefono'               => new sfValidatorInteger(array('required' => false),
  array('invalid' => 'Datos Inválidos, solo números.')),
));
```

Figura 4.3 Fragmento de código de una validación de un formulario.

4.4. Pruebas.

La fase de pruebas es una de las más costosas del ciclo de vida del software. En sentido estricto, deben realizarse pruebas a todos los artefactos generados durante la construcción de un producto, lo que incluye especificaciones de requisitos, casos de uso, diagramas de diversos tipos y, por supuesto, el código fuente y el resto de productos que forman parte de la aplicación. En la investigación se realizaron Revisiones a los documentos Modelo de Negocio, Especificación de Requisitos, Modelo de Sistema, entre otros, además Pruebas Funcionales, para éstas se diseñaron casos de pruebas a través de la descripción de los casos de usos. En el proceso de pruebas se detectaron varias No Conformidades, las cuales fueron corregidas y gestionadas correctamente.

El siguiente es un ejemplo de un Caso de Prueba realizado al Caso de Uso Realizar Solicitud.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo central
Crear	EC 1.1: Insertar los datos	Se deben insertar los datos correctamente.	1. Se registra el usuario como jefe de proyecto en el sistema.

Solicitud.	correctamente.		2. El jefe de proyecto selecciona la opción del menú Realizar Solicitud. 3. Introduce los datos correctamente.
	EC 1.2: Existencia de algún campo requerido vacío o incorrecto.	Se deben dejar campos vacíos o introducir datos incorrectos.	1. Se registra el usuario como jefe de proyecto en el sistema. 2. El jefe de proyecto selecciona la opción del menú Realizar Solicitud. 3. Introduce los datos incorrectamente o deja algún campo vacío.

A continuación un ejemplo de la matriz de datos del Caso de Uso Realizar Solicitud.

ID del escenario	Escenario	V1	V 2	V 3	V4	V5	V6	Respuesta del sistema	Resultado de la prueba
EC 1.1	Insertar los datos correctamente.	*v*	*v*	*v*				El sistema no muestra mensajes de error ya que no existen variables inválidas.	Satisfactorio.
EC 1.2	Existencia de algún campo requerido vacío o incorrecto.	*v*	*v*	*v*	*v*	*i*	*v*	El sistema muestra un mensaje de error ya que existe una variable inválida.	Satisfactorio.

4.5. Interfaces principales de la aplicación.

A continuación se muestran algunas pantallas de la aplicación:

INTERFAZ DE AUTENTICACIÓN

Figura 4.4 Interfaz de autenticación.

INTERFAZ PRINCIPAL

Figura 4.5 Interfaz principal.

INTERFAZ CREAR SOLICITUD

Figura 4.6 Interfaz Crear Solicitud.

4.6. Conclusiones del Capítulo.

Esta etapa de desarrollo se caracteriza por resultados ya visibles para los clientes y gratificantes para los desarrolladores, ya que queda implementada la aplicación con las principales funcionalidades que se definieron para la iteración del producto.

En este capítulo se elaboraron los diagramas de componentes teniéndose confeccionada completamente la propuesta que trae este trabajo. Además se brindó una breve descripción de algunos métodos implementados para obtener un mejor entendimiento de los mismos. Se citó un ejemplo de una validación realizada que permite detectar y solucionar los errores identificados así como la realización de pruebas de caja negra al software.

Conclusiones

Una vez culminado el trabajo es posible afirmar que se les dio cumplimiento a los objetivos trazados para el mismo:

- El desarrollo de la herramienta informática, creada a partir de la necesidad de automatizar los procesos que se llevan a cabo en el laboratorio de calidad de la facultad 6, permitirá elevar la eficiencia en el proceso de revisión de los productos. La implementación de esta propuesta, posibilitará dar respuesta a las solicitudes de los clientes en el menor tiempo posible.

- El sistema fue implementado utilizando herramientas, lenguajes y tecnologías propuestas por el grupo de desarrollo, en su mayoría distribuidas bajo licencias de software libre en correspondencia con las políticas de la Universidad y del país.

- Se diseñaron casos de pruebas basados en los casos de uso del sistema, además de realizar pruebas de caja negra para verificar la eficacia de la aplicación.

Recomendaciones

Después de haber logrado los objetivos que se trazaron al principio de este trabajo y como el producto informático se encuentra en su primera versión se plantea las siguientes recomendaciones:

- Implementar una nueva versión que incluya la automatización de las respuestas a las No Conformidades.
- Seguir realizando investigaciones sobre este tipo de aplicaciones para realizar mejoras en futuras versiones.
- Aplicar esta herramienta en otros laboratorios de calidad de otras facultades con el objetivo de generalizar la propuesta para elevar la eficiencia de estos procesos y evaluar las mejoras posibles para una nueva versión.

Referencias Bibliográficas

1. **IEEE.** Computer dictionary. 1990. 610.

2. **Vázquez, Roberto Hugo.** Taller de Calidad de Software. [En línea] 26 de 3 de 2006. [Citado el: 26 de 11 de 2009.] http://gridtics.frm.utn.edu.ar/docs/introduccion%20a%20la%20calidad%20de%20software%20Vazquez.pdf

3. **Vázquez, Roberto Hugo.** Taller de Calidad de Software. [En línea] 26 de 3 de 2006. [Citado el: 26 de 11 de 2009.] http://gridtics.frm.utn.edu.ar/docs/introduccion%20a%20la%20calidad%20de%20software%20Vazquez.pdf

4. **Pressman, Roberto S.** Ingeniería de Software. Un enfoque práctico. 5ta edición 2002. [En línea] [Citado el: 5 de 12 de 2009.] http://bibliodoc.uci.cu/pdf/reg02689.pdf

5. **Software, Departamento de Ingeniería de.** Pruebas,Ingeniería de Software II. [En línea] [Citado el: 5 de 12 de 2009.] http://eva.uci.cu

6. ISO.9000. 2000

7. **Jacobson, I y G, Booch.** El proceso unificado de desarrollo de software. La Habana : Félix Varela, 2004.

8. NetBeans. [En línea] http://www.netbeans.org/index_es.html

9. PHP: News Archive - 2007. 2009. [Disponible en: http://www.php.net/archive/2007.php 10954

10. **Fabien Potencier, Francois Zaninotto.** Symfony la guía definitiva. 2008. ISBN-13.

11. **Febe, A.** Utilización del Patrón Modelo Vista Controlador. 2006.

12. **C, B Reynoso.** Introducción a la Arquitectura de Software. Universidad de Buenos Aires. : s.n., 2004.

13. **Grady Booch, James Rumbaugh, Ivar Jacobson.** El Proceso Unificado de Desarrollo de Software. 1999. Vol 1.

14. Ayuda Extendida del Rational Unified Process. 2003.

15. **González, Anaisa Hernández**. [En línea] 6-7 de 2004. [Citado el: 1 de abril de 2010.] http://redalyc.uaemex.mx/pdf/615/61570402.pdf

16. **Jacobson; Booch**, et al. El Proceso Unificado de Desarrollo, 2000.

17. Profitek. [Online] 1 2, 2009 [Cited: 14 4, 2010.], de http://msdn.microsoft.com/es--es/library/aa291591%28VS.71%29.aspx

Bibliografía

1. Ayuda Extendida del Rational Unified Process. 2003.

2. ISO.9000. 2000.

3. **Fabien Potencier, Francois Zaninotto.** Symfony la guía definitiva. 2008. ISBN-13.

4. **Febe, A.** Utilización del Patrón Modelo Vista Controlador. 2006.

5. **Grady Booch, James Rumbaugh, Ivar Jacobson.** El Proceso Unificado de Desarrollo de Software. 1999. Vol 1.

6. **IEEE.** Computer dictionary. 1990. 610.

7. **Jacobson, I and G, Booch.** El proceso unificado de desarrollo de software. La Habana : Félix Varela, 2004.

8. **Pressman, Roberto S.** Ingeniería de Software. Un enfoque práctico. 5ta edición 2002. [Online] [Cited: 12 5, 2009.] http://bibliodoc.uci.cu/pdf/reg02689.pdf

9. **Software, Departamento de Ingeniería de.** Pruebas,Ingeniería de Software II. [Online] [Cited: 12 5, 2009.] http://eva.uci.cu

10. **Vázquez, Roberto Hugo.** Taller de Calidad de Software. [Online] 3 26, 2006. [Cited: 11 26, 2009.] http://gridtics.frm.utn.edu.ar/docs/introduccion%20a%20la%20calidad%20de%20software%20Vazquez.pdf

11. **Reynoso, C B.** Introducción a la Arquitectura de Software. Universidad de Buenos Aires : s.n., 2004.

12. Desarrollo Web. [Online] 1 23, 2009. http://www.desarrolloweb.com/articulos/25.php

13. Profitek. [Online] [Cited: 1 2, 2009.] http://www.profitek.com.co/productos/signo.htm

14. Profitek. [Online] 1 2, 2007. [Cited: 12 3, 2009.]
http://www.kmkey.com/productos/software_gestion_calidad.

15. **Borrero, Martha Nieves and Rodríguez, Asnier Góngora.** Herramienta Informática para automatizar los procesos en el laboratorio de Calidad de Software: Módulo Gestión de las No Conformidades. La Habana : s.n., 2007.

16. Sun Mycrosystem. [Online] 2008-2009. [Cited: 12 13, 2009.] http://dev.mysql.com/doc/refman/5.0/es/features.html

17. **Jacobson and Booch.** El Proceso Unificado de Desarrollo de Software. 2000.

18. **Orallo.** El Lenguaje Unificado de Modelado(UML). [Online] [Cited: 1 26, 2010.] http://www.disca.upv.es/enheror/pdf/ActaUML.PDF

19. New Releases and Feature Enhancements of Visual Paragim Products. Disponible en: http://www.visual-paradigm.com/product/news.jsp#VPUML50 5199

20. PHP: News Archive - 2007. 2009. [Disponible en: http://www.php.net/archive/2007.php 10954

21. **Concepción, Yadicza López and Barroso, Keidis Martínez.** Herramienta Informática para automatizar los procesos en el laboratorio de Calidad de Software: Módulo Seguimiento de los Recursos. La Habana : s.n., 2007.

22. **González, Anaisa Hernández.** [En línea] 6-7 de 2004. [Citado el: 1 de abril de 2010.] http://redalyc.uaemex.mx/pdf/615/61570402.pdf

23. **Jacobson; Booch**, et al. El Proceso Unificado de Desarrollo, 2000.

24. Profitek. [Online] 1 2, 2009 [Cited: 14 4, 2010.], de http://msdn.microsoft.com/es--es/library/aa291591%28VS.71%29.aspx

Anexos

Anexo 1: Descripción Textual Caso de Uso "Realizar Solicitud".

Caso de Uso:	Realizar Solicitud Prueba.
Actores:	Jefe de Proyecto.
Resumen:	El Caso de Uso se inicia cuando el jefe de proyecto realiza la solicitud de revisión de su producto, se muestra la interfaz correspondiente para llenar la solicitud y el caso de uso termina cuando introduce todos los datos.
Precondiciones:	Existencia de proyectos para revisión.
Referencias	RF 3
Prioridad	Crítico.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso se inicia cuando el jefe de proyecto realiza la solicitud de revisión de su producto.	2. El sistema muestra la interfaz correspondiente para la creación de una solicitud con los siguientes datos: • Proyecto. • Módulos. • Artefactos. • Nombre del solicitante. • Dirección de correo del solicitante. • Teléfono del solicitante.
3. El jefe de proyecto introduce los datos necesarios para crear una solicitud.	4. Valida que los datos estén correctos.
	5. Crea una solicitud.
	6. El sistema da la opción de crear otra solicitud.
7. El jefe de proyecto indica que desea realizar otra solicitud	8. El sistema va a la acción 1 de esta Sección.
Prototipo de Interfaz	

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	4.1. En caso que los datos no estén correctos muestra un mensaje de error "Datos Incorrectos".
7.1 Si el jefe de proyecto no desea realizar otra solicitud finaliza el caso de uso.	

Prototipo de Interfaz

Poscondiciones	Es creada la solicitud.

Glosario de Términos

1. **AJAX:** Es una técnica de desarrollo web para crear aplicaciones interactivas.
2. **API's:** Una interfaz de programación de aplicaciones o API (del inglés application programming interface).
3. **Apache:** Es un software (libre) servidor HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etcétera), Windows y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.
4. **Calisoft:** Centro para la Excelencia en el Desarrollo de Proyectos Tecnológicos.
5. **CMMI:** Capability Maturity Model Integration. Es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.
6. **C++, CORBA IDL, PHP, Ada y Python:** Lenguajes de programación.
7. **Diagrama:** Representación gráfica en la que se muestran las relaciones entre las diferentes partes de un conjunto o sistema.
8. **Entidades:** Objetos concretos o abstractos que presentan interés para el sistema.
9. **Hardware:** Componentes físicos que constituyen las Computadoras y demás dispositivos periféricos.
10. **HTML:** Acrónimo inglés de Hyper Text Markup Language (lenguaje de marcación de hipertexto), es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web. Gracias a Internet y a los navegadores del tipo Explorer o Netscape, el HTML se ha convertido en uno de los formatos más populares que existen para la construcción de documentos.
11. **ISO:** Organización Internacional para la Estandarización.
12. **Java Script:** Es un lenguaje interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas Web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C. Al contrario que Java, Javascript no es un lenguaje orientado a objetos propiamente dicho, ya que no dispone de herencia, es más bien un lenguaje basado en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.
13. **PHP:** Es un lenguaje de programación usado generalmente para la creación de contenido para sitios web.
14. **Rollback's:** Es una operación que devuelve a la base de datos a algún estado previo.

15. **SIGNCR:** Sistema Informático para la Gestión de No Conformidades y Recursos del grupo de calidad de la facultad 6.
16. **SSL:** Es un protocolo de seguridad.
17. **URL:** Localizador uniforme de recursos, más comúnmente denominado URL (sigla en inglés de uniform resource locator).