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

Tema: Sistema de Gestión Integral de la Calidad de Software para el Centro de Gobierno Electrónico.

Autores: Anabel Fé León Mendoza
Ebrik René López Ramirez

Tutor: Ing. Raúl Velázquez Alvarez

Ciudad de La Habana, Junio de 201

AGRADECIMIENTOS

Anabel Fé León Mendoza

Para llevar a cabo mis sueños fueron muchas las personas que influyeron a lo largo de mi carrera.

Quiero agradecer en primer lugar a mi mamá Lina Mendoza Rodríguez por ser mi guía, mi consejera por siempre apoyarme y llorar juntas cuando salía mal en las pruebas. Por su paciencia infinita, porque ha sido madre y padre y a pesar de ello ha sabido llevarme por el buen camino, porque sin ti hoy mis sueños no se estuviesen realizando.

A mi hermana Rosabel por ser la mejor hermana del mundo y apoyarme siempre, gracias por estar siempre a mi lado y pasar las madrugadas estudiando y regañándome para poder salir bien en las pruebas. Gracias por ser mi doble y compartir conmigo los momentos buenos y malos.

A mi papá Alejandro porque a pesar de la distancia, siempre tuve su apoyo incondicional.

A mi tía Sara por ser mi segunda mamá y darme siempre muy buenos consejos para poder seguir adelante con mis sueños.

A mi prima Sailen por las largas noches viendo novelas coreanas.

A mi novio Leandro por siempre estar a mi lado cuando más falta me hacía, por apoyarme y darme todo el amor del mundo, porque a pesar de estos 6 años juntos, nunca dejaste de confiar en mí y demostrarme que con amor podemos llegar muy lejos. Te amo.

A mi suegra Yoyi por quererme como una hija y dejarme formar parte de su familia.

A mis compañeras de apartamento Diana, Lorena, Rosabel, Grettel y Yaineris porque a pesar de no vivir juntas pasamos largas noches de tertulias y nos apoyábamos siempre las unas a las otras.

A mi grupo 3504 porque de cada uno aprendí, pero tengo que resaltar a Sandy, Cesar, Maceo, Carlos, Yasmani, Betty, Diana, Lorena, Yiselly y Rosabel por siempre contar con su apoyo incondicional y en especial a mi compañero de tesis Ebrik que a pesar de sacarme del paso algunas veces, siempre enfrentamos las dificultades juntos para poder realizar nuestros sueños.

A mi tutor Raúl por formar parte de nuestro equipo, por todo su apoyo y toda la ayuda que nos ha brindado para llevar a cabo este trabajo de diploma.

A Hernán por su ayuda incondicional.

A todas aquellas personas que de alguna forma u otra hicieron posible este momento, mil gracias.

Ebrik René López Ramirez

Agradecer ante todo a Dios por ser mi pie de apoyo para lograr todas mis metas.

A mi papá por cada gota de sudor que ha derramado para que yo pueda cumplir mis sueños, por la confianza y el apoyo.

A mi mamá, por su apoyo incondicional, por ayudarme a levantar después de cada tropiezo y por los consejos que tanto me han ayudado a seguir siempre adelante.

A mi hermana, por todo su apoyo y cariño.

A mi novia, por todo el tiempo que ha estado a mi lado, por la felicidad que me ha dado y por su apoyo en todo momento.

A mis abuelas Olga y Oli, a mi abuelo Roberto, a mis tíos, lo mismo los de aquí de la Habana como los de Fomento, muchas gracias por todo su ayuda y apoyo. Agradecer a toda mi familia en general.

Agradecer a todos mis amigos de la Universidad, en especial los de mi grupo: principalmente Anabel la jimagua mi querida compañera de tesis por todo el trabajo que hemos pasado para lograr este sueño, mis compañeros de apartamento el loquillo Arian y el loco Sarmiento por haber compartido tanto con ustedes en estos 5 años, agradecer también a Anabel Valiente, Rosabel la otra jimagua, Betty, Cesar, Diana, Edisvel, Maceo, Laura, Lorena, Michel, Oridalmis, Sandy, Yasmani, Yiselly, agradecer a mis grandes amigos Rojas y Luis que hemos compartido mucho este año.

Dar gracias a mi tutor Raúl, por todo su apoyo y toda la ayuda que nos ha brindado para llevar a cabo este trabajo de diploma.

A todas las personas que he conocido y compartido en estos 5 años y que de una forma u otra me ayudaron y me apoyaron, sin ellos no hubiera sido posible este momento, MUCHAS GRACIAS...

DEDICATORIA

Anabel Fé León Mendoza

Dedico el presente trabajo de diploma a mi mamá por ser la mejor mamá del mundo y darme las fuerzas para poder seguir adelante con mis sueños.

A mi hermana porque sin su apoyo incondicional hoy no estaría aquí

.

Ebrik René López Ramirez

A mi papá.

A mi mamá.

A mi hermana.

A mi novia.

A mi niño que está por nacer.

A mis abuelas Oli y Olga, a mi abuelo Roberto, a todos mis tíos, a toda mi familia en general.

A mis amigos de la universidad.

A todas esas personas que hicieron posible este momento.

Declaración Jurada de Autoría

Declaramos que somos autores del presente trabajo de diploma y reconocemos a la Universidad de la Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmo la presente a los——— días del mes de——————— del año————————

Anabel Fé León Mendoza

_____

Firma del Autor

Ebrik René López Ramirez

_____

Firma del Autor

Ing. Raúl Velázquez Alvarez

_____

Firma del Tutor

RESUMEN

El presente trabajo de diploma está dedicado al desarrollo del Sistema de Gestión Integral de la Calidad de Software para el Centro de Gobierno Electrónico (CEGEL). El mismo tiene como objetivo contribuir a la mejora del proceso de gestión de la calidad, disminuyendo el tiempo de recolección y análisis de los datos del área.

Para ello se realiza el estudio del estado del arte de la temática a tratar, identificando los conceptos principales de la investigación así como la selección de la metodología, lenguajes y herramientas utilizadas como apoyo para darle solución al problema planteado. Además, se hace un análisis de los sistemas informáticos que miden la gestión de la calidad de software, tanto a nivel nacional como internacional, demostrándose que estos no resuelven los problemas del Grupo de Calidad del Centro de Gobierno Electrónico. Se define la arquitectura del sistema así como un conjunto de artefactos necesarios para lograr la implementación. Para evitar inconsistencias durante el desarrollo y cumplir con las expectativas del cliente se validan los requisitos y el diseño. Por último se realizan pruebas al sistema que verifican su correcto funcionamiento y se demuestra que la solución propuesta resuelve las necesidades del Grupo de Calidad del Centro de Gobierno Electrónico.

Palabras Claves

Calidad de Software, Gestión de la Calidad, Sistema de Gestión Integral.

ÍNDICE

Introducción 1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 6
1.1 Introducción 6
1.2 Gestión de la calidad de Software 6
1.3 Escenario actual del proceso 8
1.4 Valoración del estado del arte 8
1.5 Metodología, Herramientas y Tecnologías 10
1.5.1 Selección de la metodología 10
1.5.2 Selección del tipo de aplicación 11
1.5.3 Lenguaje de modelado 12
1.5.4 Herramientas utilizadas para el modelado 12
1.5.5 Selección del entorno de desarrollo 12
1.5.6 Marco de trabajo 13
1.5.7 Lenguajes de programación 14
1.5.8 Herramientas para base de datos 16
1.5.9 Servidor web 16
1.6 Patrones 17
1.6.1 Patrón de arquitectura 17
1.6.2 Patrones de diseño 17
1.7 Métricas 18
1.7.1 Métricas para requisitos 18
1.7.2 Métricas para el diseño 20
1.8 Pruebas 22
1.8.1 Método de prueba de Caja Negra 22
1.8.2 Método de prueba de Caja Blanca 23
1.8.3 Pruebas de aceptación 23
1.9 Conclusiones parciales 23
CAPÍTULO 2: PLANIFICACIÓN, DISEÑO E IMPLEMENTACIÓN DE LA PROPUESTA DE SOLUCIÓN 25
2.1 Introducción 25
2.2 Propuesta de solución 25
2.3 Requisitos 26
2.3.1 Técnicas para la captura de requisitos 26
2.3.2 Requisitos no Funcionales 32
2.3.3 Actores del sistema 34
2.4 Planificación 35

2.4.1 Historias de Usuario 35
2.4.2 Desarrollo de iteraciones 39
2.5 Diseño de la solución 40
2.5.1 Arquitectura del sistema 40
2.5.2 Tarjetas Clases Responsabilidad Colaborador (CRC) 44
2.5.3 Modelo de datos 45
2.5.4 Patrones de diseño 46
2.6 Codificación de la solución 48
2.6.1 Estándares de codificación 49
2.7 Conclusiones parciales 51
CAPÍTULO 3: VALIDACIÓN DE LA INVESTIGACIÓN 52
3.1 Introducción 52
3.2 Validación de los requisitos 52
3.2.1 Métricas para requisitos 52
Aplicación de la métrica Estabilidad de requisitos: 52
3.3 Validación del diseño 53
3.4 Verificación del sistema 56
3.4.1 Nivel de Unidad 56
3.4.1.2 Método de pruebas de caja negra 56
3.4.1.3 Método de pruebas de caja Blanca 58
3.4.2 Nivel de Integración 61
3.5 Validación del sistema 61
3.5.1 Nivel de aceptación 61
3.6 Validación de las variables de la investigación 63
3.7 Conclusiones parciales 64
CONCLUSIONES GENERALES 65
RECOMENDACIONES 66
Bibliografía 67

Índice de Figura

Fig. 1 Esquema de la arquitectura 41
Fig. 2 Patrón MVC 42
Fig. 3 Estructura de Clases 43
Fig. 4 Modelo de datos 46
Fig. 5 Patrón de llave subrogada 47
Fig. 6 Resultado de la métrica RC 54
Fig. 7 Resultado de la métrica TOC 55
Fig. 8 Caso de Prueba Adicionar Proyecto 57
Fig. 9 Errores detectados en las pruebas funcionales 57
Fig. 10 Código fuente del método updateAction 59
Fig. 11 Grafo de flujo del método updateAction 59

Índice de Tabla

Tabla 1 Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica 21
Tabla 2 Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica 22
Tabla 3 Historia de Usuario Gestionar roles 36
Tabla 4 Historia de Usuario Gestionar tarea 37
Tabla 5 Historia de Usuario Gestionar Nomencladores 38
Tabla 6 Plan de duración de las iteraciones 39
Tabla 7 Distribución del plan de iteraciones 40
Tabla 8 Tarjeta CRC Módulo 45
Tabla 9 Tarjeta CRC Pregunta 45
Tabla 10 Tarjeta CRC Plan de revisiones 45
Tabla 11 Tarea de Ingeniería 49
Tabla 12 Caso de prueba para el camino 4 60
Tabla 13 PA Autenticar usuario 62
Tabla 14 Validación de las variables 63

Introducción

En la actualidad, el avance de las tecnologías de la Informática y las Telecomunicaciones (TIC) han posibilitado el desarrollo de las industrias de software para la informatización de la sociedad, convirtiéndose estas en el núcleo central de las transformaciones que afronta la economía, trayendo como consecuencia un aumento de la productividad y una mejor gestión de las empresas y organizaciones (TIC, 2015). Esto abre paso a que la informática se convierta en la ciencia clave para el progreso en casi todas las esferas de un país y la producción de software ocupe un lugar significativo en el ámbito económico mundial.

Cada día aumenta el número de organizaciones de desarrollo de software que deciden mejorar la calidad de sus productos y servicios, no obstante, existe un número de ellas que gestionan parte de sus procesos principales a través de sistemas informáticos que en muchos de los casos trabajan de manera independiente, provocando retraso en la gestión de la información de sus procesos. La gestión integral de sistemas informáticos, es una alternativa que pueden usar estas organizaciones de desarrollo de software para integrar varios procesos y ver a la organización como un solo sistema en el que todo está relacionado. Esto maximiza el valor de la información y evita realizar esfuerzos en tareas de coordinación u organización de la información, agilizando la toma de decisiones.

Cuba, independientemente de no encontrarse entre los países desarrollados incorpora el uso de las TIC en función del bienestar social. De este modo, ha realizado un gran esfuerzo por llevar la informática a cada uno de los sectores de la producción en el país, con el objetivo de lograr la informatización de la sociedad. La Universidad de las Ciencias Informáticas (UCI), es uno de los proyectos de la Revolución Cubana que tiene entre sus misiones, informatizar la sociedad, para ello ha creado un grupo de proyectos productivos cuyos resultados evidencian el cumplimiento de tales objetivos. De esta forma, surge el Centro de Gobierno Electrónico (CEGEL), constituido en la Facultad 3, el cual tiene la misión de satisfacer necesidades de clientes gubernamentales mediante el desarrollo de productos y servicios de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia, a partir de un personal calificado.

El Grupo de Calidad del centro CEGEL de la Facultad 3 fue creado con el objetivo de mejorar la calidad del proceso de desarrollo y de los productos producidos por el centro CEGEL. Este grupo tiene la responsabilidad de la planificación, supervisión, mantenimiento de registros, análisis e informes relacionados con actividades de calidad entre las que se encuentran: revisión de los artefactos de la metodología y el modelo utilizado, para verificar su ajuste al proceso de desarrollo de software definido y la evaluación de la satisfacción de los equipos de desarrollo, a partir de las

encuestas realizadas a cada uno de sus miembros, logrando así la calidad de los productos en el centro.

Para ello se han realizado investigaciones referentes a la satisfacción del cliente, revisiones y aseguramiento de las características de la calidad, sustentándola con 3 sistemas que tributan a la gestión de la calidad en el proceso de desarrollo de los proyectos del centro, los cuales son:

- ✓ Herramienta para evaluar la satisfacción de los clientes internos en el proceso de desarrollo de software de los proyectos del centro CEGEL de la Facultad 3.
- ✓ Herramienta para la gestión del proceso de revisión de aseguramiento de la calidad a proceso y producto en el Grupo de Calidad de software del centro CEGEL.
- ✓ Herramienta para evaluar el grado de implementación de las características de calidad en el proceso de desarrollo de los sistemas informáticos del centro CEGEL.

A estos sistemas acceden prácticamente los mismos usuarios, ya que intervienen en la ejecución de los distintos procesos que en estos se gestionan. Algunas funcionalidades son similares y retrasan las actividades a desarrollar; esto resulta tedioso, pues cada vez que se utilizan tienen que acceder a cada uno por separado. Cada uno funciona de forma aislada y utilizan bases de datos diferentes en las que existe información repetida. Las tecnologías que soportan algunos son diferentes, con estilos de diseño e interfaces, arquitectura de información y tipografía distintas, que afectan la usabilidad. Este conjunto de elementos incide negativamente en la ejecución del proceso de gestión de la calidad como un todo, pues se dificulta el acceso a la información del área ya que no se encuentra centralizada, percibiéndose un aumento en el tiempo de recolección y análisis de los datos.

A partir de la problemática identificada se plantea el siguiente **problema de la investigación:** ¿Cómo contribuir a la mejora del proceso de Gestión de la Calidad de Software del Centro de Gobierno Electrónico de manera que se facilite la ejecución de las actividades del mismo y disminuya el tiempo de recolección y análisis de los datos del área?

Partiendo del problema anteriormente planteado se ha determinado como **objeto de estudio:** Proceso de Gestión de la Calidad de Software y como **objetivo general:** desarrollar un sistema de gestión integral de la calidad de software, para contribuir a la mejora del proceso de gestión de la calidad del Centro de Gobierno Electrónico, de manera que se facilite la ejecución de las actividades del mismo y disminuya el tiempo de recolección y análisis de los datos del área.

Tomando como **campo de acción**: Sistemas informáticos de gestión de la calidad de software.

Se plantea como **idea a defender:** con el desarrollo de un sistema de gestión integral de la calidad de software se contribuye a la mejora del proceso de gestión de la calidad del Centro de Gobierno Electrónico, facilitando la ejecución de las actividades del mismo y disminuyendo el tiempo de recolección y análisis de los datos del área.

Del objetivo general se desglosan los siguientes **objetivos específicos**:

- ✓ Elaborar el marco teórico para sustentar las bases de la investigación.
- ✓ Desarrollar la propuesta de solución para darle cumplimiento al objetivo general.
- ✓ Realizar la validación de la investigación para demostrar que la propuesta de solución resuelve el problema planteado.

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

- ✓ Realización de un estudio del estado del arte de la temática en Cuba y el mundo para la definición de los principales conceptos de la investigación y selección de la metodología, herramientas y tecnologías más adecuadas para el desarrollo de la solución.
- ✓ Realización de la captura de los requisitos para obtener las especificaciones funcionales y no funcionales que debe cumplir el sistema.
- ✓ Realización del diseño del sistema para obtener la modelación necesaria como base para la implementación de los requisitos definidos.
- ✓ Definición de la base de datos para mejorar la consistencia de los datos y prever que no exista información repetida en el sistema.
- ✓ Diseño de los casos de prueba para la realización de las pruebas al sistema.
- ✓ Realización de pruebas de caja negra y caja blanca para comprobar el correcto funcionamiento del sistema.

Los métodos científicos utilizados para realizar la investigación se citan a continuación:

Métodos Teóricos:

- ✓ Analítico-Sintético: permite el procesamiento de la información para arribar a conclusiones prácticas y teóricas de la investigación. Mediante este método se realiza un análisis de los documentos, las publicaciones, bibliografías y en general toda la información relacionada con los sistemas de gestión integral de la calidad de software para reflejar los elementos más importantes y necesarios en el desarrollo de la solución.

- ✓ Histórico-lógico: se utilizó durante la realización del estudio de los diferentes sistemas de gestión de la calidad, permitiendo evaluar el desarrollo de las diferentes herramientas de gestión de la calidad de software.
- ✓ Modelación: este método se encarga de realizar una reproducción simplificada de la realidad, abstracciones, con el objetivo de explicarla y comprender mejor los procesos del negocio de la presente investigación. Se utiliza en el diseño del sistema mediante el esbozo de los diferentes diagramas definidos en la metodología escogida, permitiendo así un mejor entendimiento de las funcionalidades de la aplicación.

Métodos Empíricos:

- ✓ Medición: permite obtener a partir del uso de las métricas para validar los requisitos y el diseño, la información numérica que se necesita para comprobar la correcta elaboración de los mismos. Así como obtener resultados estadísticos de la realización de las pruebas al sistema.
- ✓ Entrevista: permite obtener información valiosa sobre las necesidades del cliente para identificar los requisitos en el desarrollo de software, obteniendo información acerca de la situación existente en el Grupo de Calidad del Centro de Gobierno Electrónico y definir los requisitos que debe tener el sistema.

El presente trabajo de diploma queda estructurado por 3 capítulos.

Capítulo 1. Fundamentación teórica: en este capítulo se lleva a cabo el estudio del estado del arte donde se realiza un análisis de las diferentes herramientas para llevar a cabo la realización de un sistema de gestión integral para el Grupo de Calidad del Centro de Gobierno Electrónico. Se enunciaron los principales conceptos relacionados con el tema, así como una selección de la metodología, lenguajes y herramientas usadas como apoyo para darle solución al problema planteado.

Capítulo 2. Planificación, diseño e implementación de la propuesta de solución: se especifican los requisitos funcionales y no funcionales, teniendo en cuenta las restricciones o políticas a cumplir por el sistema. Además se realiza la descripción del comportamiento del sistema a través de las Historias de Usuarios. Como parte del diseño se define el patrón de arquitectura y se identifican los patrones de diseño que se van a utilizar, posteriormente se confeccionan las Tarjetas CRC (Clase-Responsabilidad-Colaborador) que contienen el nombre de las clases, las responsabilidades y sus colaboradores. Luego se diseña la base de datos, se elaboran las Tareas de Ingeniería y se determinan los estándares de codificación para llevar a cabo la implementación del sistema.

Capítulo 3. Validación de la investigación: en este capítulo se realizan la validación de los requisitos y el diseño, así como pruebas para la verificación y validación del sistema, para confirmar a través de estas últimas si se satisfacen las necesidades del cliente. Por último se validan las variables de la investigación para comprobar que el objetivo resuelve el problema planteado.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se exponen los conceptos fundamentales utilizados a lo largo de la investigación. Además se realiza la selección de la metodología, entorno de desarrollo, gestor de base de datos, técnicas y herramientas a utilizar en la implementación.

1.2 Gestión de la calidad de Software

Según la Norma ISO 9000 gestión de la calidad "*es un conjunto de actividades de la función general de la dirección que determina la calidad, los objetivos y las responsabilidades y se implanta por medios tales como la planificación de la calidad, el control de la calidad, el aseguramiento de la calidad y la mejora de la calidad, en el marco del sistema de calidad"* (ISO-9000, 2000).

Este concepto enfocado al software no es más que el conjunto de actividades organizadas para lograr la calidad de los sistemas informáticos durante todo el proceso de desarrollo. Se llevan a cabo a través de revisiones planificadas, incluyendo además en el contexto de esta investigación, el aseguramiento de las características de calidad para evaluar el grado de implementación de estas durante todo el proceso de desarrollo del producto. Por último se tiene en cuenta la evaluación de la satisfacción de los clientes para la mejora continua del proceso en la organización.

La ISO/IEC DEC 9126 define calidad de software como la "*totalidad de características de un producto de software que tienen como habilidad, satisfacer necesidades explícitas o implícitas"*. Roger S. Pressman la define como "*concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente"* (Pressman, 2010).

Asociado al concepto de calidad del software surgen términos como el aseguramiento de la calidad, el cual se puede definir como "*el esfuerzo total para plantear, organizar, dirigir y controlar la calidad en un sistema de producción con el objetivo de dar al cliente productos con la calidad adecuada"*. Es simplemente asegurar que la calidad sea lo que debe ser (Lopez, 2000). Según la Norma ISO 9000, el aseguramiento de la calidad *"es la parte de la gestión de la calidad orientada a proporcionar confianza en que se cumplirán los requisitos de la calidad"* (ISO-9000, 2000). Roger Pressman lo define como el *"conjunto de actividades planificadas y sistemáticas necesarias para aportar la*

seguridad en que el producto (software) satisfaga los requisitos dados con calidad" (Pressman, 2000).

Por su parte el Modelo Integrado de Capacidad y Madurez (CMMI por sus siglas en inglés) en su nivel 2 de madurez, define el área Aseguramiento de la Calidad del Proceso y del Producto (PPQA por sus siglas en inglés), que tiene como propósito *"proporcionar a los diferentes equipos y a la gerencia una visión objetiva de los procesos y productos asociados"* (software, 2010). Esto se garantiza a través de las revisiones de aseguramiento de la calidad durante todo el proceso de desarrollo para comprobar que los procesos definidos están siendo respetados en la organización, así como poder detectar deficiencias en la forma de trabajar establecida (Magdalena, 2010).

Luego de haber analizado todos estos criterios, es necesario introducir además en el contexto de esta investigación al aseguramiento de las características de calidad. Según el criterio de los autores del presente trabajo, tomando como base lo anteriormente explicado, no es más que el diseño planificado de acciones o actividades a realizar a lo largo del ciclo de vida de software que se requieren para garantizar las características de calidad en el producto final.

Otro elemento importante es la medición de la satisfacción no solo de los clientes finales sino también de los clientes internos, en este caso, los equipos de desarrollo. Es por eso que Jorge Pereiro plantea lo siguiente: *"La satisfacción de las necesidades y expectativas del cliente constituye el elemento más importante de la gestión de la calidad y la base del éxito de una empresa".* Enfatiza la necesidad que tienen las organizaciones de conocer la satisfacción de sus clientes, desarrollando sistemas para su medición y creando modelos de respuesta inmediata ante la posible insatisfacción (Pereiro, 2001).

Después de un análisis de los conceptos antes mencionados los autores de la presente investigación llegaron a la conclusión, que todo este conjunto de elementos en el proceso de desarrollo de software se ven de manera relacionada. Por lo que cabe resaltar que un sistema de gestión integral de la calidad de software permite gestionar los procesos de gestión de la calidad de forma integrada, a través de un solo sistema.

A partir del análisis de los diferentes conceptos referentes al proceso de gestión de la calidad de software, es necesario conocer la situación actual de los diferentes procesos que se llevan a cabo en el Grupo de Calidad del centro CEGEL para contribuir así a la mejora del mismo.

1.3 Escenario actual del proceso

Actualmente en el Centro de Gobierno Electrónico para la evaluación de la satisfacción del cliente, se utiliza el sistema ESCISoft que permite evaluar la satisfacción de los clientes internos en el proceso de desarrollo de software a través de encuestas, para facilitar la identificación de causas de insatisfacción como apoyo a la toma de decisiones administrativas en dicho centro. Además para detectar no conformidades y realizar revisiones a los proyectos del centro se utiliza la herramienta SGPRC, encargada de gestionar los procesos de revisiones de aseguramiento de la calidad del proceso y del producto, facilitándole a los revisores detectar errores a través de listas de chequeo para su posterior análisis. Permite además al Asesor de la calidad realizar la planificación anual de revisiones de adherencia a proceso y producto, determinando cuándo le corresponde a cada proyecto la revisión y para ello se le envía una notificación de revisión, la cual se hace por medio del correo electrónico o de forma personal. Por otra parte, la herramienta HEGICC evalúa el grado de implementación de las características de calidad, con el objetivo de chequear el avance del grado de implementación por características y por disciplinas de los proyectos del centro.

Para el desarrollo de la investigación fue necesario realizar un estudio de algunos sistemas informáticos que dan muestra de la utilización de los sistemas de gestión de la calidad de software, para determinar qué elementos pueden ser utilizados en la implementación del sistema que se desea obtener con esta investigación.

1.4 Valoración del estado del arte

Se comienza el análisis de los sistemas similares en el marco internacional entre los cuales se destacan los siguientes sistemas informáticos de referencia:

Herramienta para certificar la madurez de los procesos que requiere CMMI nivel 2: en la Facultad de Informática de la Universidad Nacional de la Plata, se crea una herramienta para certificar la madurez de los procesos que requiere CMMI nivel 2, ayudando para ello al personal que realiza el aseguramiento de la calidad. El objetivo principal de la misma es llevar el seguimiento de las auditorías y del registro y tratamiento de las no conformidades requeridos por el Área de PPQA. Entre las funcionalidades que presenta se encuentra generar y escalar no conformidades, recordar al usuario las auditorías y no conformidades registradas, crear proyectos a partir de plantillas predefinidas, agregar y registrar auditorías, configuración general para el seguimiento y control de cumplimiento de auditorías y verificar el cumplimiento de las tareas de una auditoría (Gabriel Vargas, 2009).

KMKey Quality (Knowledge Management Key): 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, OHSAS 18001, etc, o de una combinación de los mismos, facilitando la gestión de un sistema integrado. Mediante KMKey Quality se gestiona y mantiene 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 y preventivas, reclamaciones, auditorías, quejas, evaluaciones, indicadores, entre otros. Todo ello adaptado al enfoque propio de la organización. Este sistema realiza encuestas de satisfacción de los usuarios y de los servicios, lo cual permite acceder a los registros de las mediciones de la satisfacción del cliente y seguir su evolución, así como realizar el seguimiento de las acciones definidas en función del análisis de los resultados obtenidos (Kmkey, 2015).

Sistema Integrado de Gestión: procedimiento para medir la satisfacción de los funcionarios de la Universidad de Caldas frente a los servicios internos de los procesos. La medición de la satisfacción se realiza por medio de la aplicación de dos encuestas. Las encuestas se administran a través del Sistema Integrado de Gestión en el módulo de encuestas institucionales. De manera aleatoria el software le asigna una de las encuestas al personal seleccionado. Con la aplicación de estos instrumentos para medir la satisfacción se permitirá evaluar las condiciones institucionales para propósitos de acreditación institucional (Hurtado, 2009).

Se relacionan a continuación los sistemas similares en el marco nacional:

Gespro (Ecosistema de software para la Dirección Integrada de Proyectos): combina el uso de una solución informática para la dirección integrada de proyectos y un sistema de formación especializada en gestión de proyectos. Este sistema permite la gestión de listas de chequeo asociadas a las revisiones de los proyectos, gestiona las no conformidades y seguimiento de las mismas, permite asignar tareas, así como notificarlas, además permite realizar las planificaciones de la calidad (Gespro, Xedro, 2003).

Herramienta para evaluar la satisfacción de los clientes internos en el proceso de desarrollo de software de los proyectos del Centro de Gobierno Electrónico de la Facultad #3: permite evaluar la satisfacción del cliente interno en el proceso de desarrollo de software de los proyectos del centro CEGEL facilitando la identificación de causas de insatisfacción como apoyo a la toma de decisiones administrativas en dicho centro (Labrada, 2013).

Herramienta para la gestión del proceso de revisión de aseguramiento de la calidad a proceso y producto del Grupo de Calidad de software del Centro de Gobierno Electrónico: permite realizar toda la gestión del proceso de revisión de PPQA en el Grupo de Calidad del centro CEGEL de los proyectos realizados en el centro. Facilita la gestión de listas de chequeo asociadas a las revisiones de los proyectos, gestiona las no conformidades y seguimiento de las mismas, permite asignar tareas, así como notificarlas, además permite realizar las planificaciones de la calidad entre otras (Orlando, 2014).

Herramienta para evaluar el grado de implementación de las características de calidad en el proceso de desarrollo de los sistemas informáticos del Centro de Gobierno Electrónico: esta herramienta permite realizar una evaluación del grado de implementación de las características de calidad durante el proceso de desarrollo de software de los sistemas informáticos del centro CEGEL, para contribuir al aseguramiento de las características de calidad en el producto final (Felix, 2014).

Los sistemas antes mencionados tienen como elemento común que algunos utilizan como técnica la aplicación de encuestas y que los cuestionarios que se emplean incluyen preguntas relacionadas con las condiciones de trabajo y con la motivación, otros por su parte presentan una pequeña parte del proceso de revisiones y del grado de implementación de las características de calidad. Estos sistemas informáticos no satisfacen las necesidades del centro CEGEL en su totalidad, porque algunos están implantados en otros países y son software propietarios por lo que no se puede acceder a ellos, además no consideran los roles para la evaluación de la satisfacción de los clientes internos, ni se tiene en cuenta la dimensión competencia técnica que constituye un elemento imprescindible a medir en el centro. Por otra parte, algunos no se aplican en entornos de desarrollo de software por lo que no están dirigidos a los mismos objetivos y necesidades que se requieren, es por ello que se hace necesario la creación de un sistema para la gestión integral de la calidad de software. A continuación se realiza un estudio de las metodologías más utilizadas en la actualidad para seleccionar la que más se ajuste a las necesidades del sistema a implementar y utilizarla como guía en el ciclo de desarrollo.

1.5 Metodología, Herramientas y Tecnologías

1.5.1 Selección de la metodología

"La metodología de desarrollo de software se encarga de elaborar estrategias; centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega.

Su principal objetivo es elevar la calidad de software a través de un mayor control sobre el proceso" (Sommerville, 2005).

Las metodologías de desarrollo se pueden dividir en dos grupos, de acuerdo a sus características y los objetivos que persiguen *Ágiles* y *Tradicionales* (Velthuis, 2003).

Metodologías tradicionales o pesadas: hacen mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, estableciendo estrictamente las actividades involucradas, los roles definidos, los artefactos que se deben producir, las herramientas y la documentación usada (RUP (Rational Unified Procces), MSF (Microsoft Solution Framework), Win-Win Spiral Model, Iconix).

Metodologías ágiles o ligeras: orientadas a la generación de código con ciclos muy cortos de desarrollo manteniendo un proceso incremental, son capaces de permitir cambios en los requisitos de último momento, además el equipo de desarrollo mantiene una comunicación constante con el cliente (Extreme Programming (XP), SCRUM, Crystal Clear, Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD)).

Después de analizar los dos grupos, se definió el uso de una metodología ligera, escogiéndose la programación extrema (XP, por sus siglas en inglés, Extreme Programming desarrollada por Kent Beck en el año 2001) porque se centra en las necesidades del cliente para lograr un producto de buena calidad en poco tiempo. Además el equipo de desarrollo presenta solo dos integrantes y la planificación del cronograma de ejecución establece poco tiempo para darle respuesta al problema en cuestión. Ofrece la posibilidad de cambiar los requisitos en cualquier momento de la vida de un proyecto, ya que es adaptable a cambios. El cliente se convierte en miembro del equipo y decide qué se implementa, además de saber el estado real y el progreso del proyecto. Puede añadir, cambiar o quitar requerimientos en cualquier momento. Se basa en una retroalimentación continuada entre el cliente y el equipo de desarrollo con una comunicación fluida entre todos los participantes como clave para el éxito en el desarrollo de software.

1.5.2 Selección del tipo de aplicación

Aplicación web

Se decide realizar una aplicación web por su facilidad para actualizarla y mantenerla, sin distribuir e instalar software a los diferentes usuarios que harán uso del sistema. Además, esta se puede ejecutar desde cualquier ordenador que tenga instalado un navegador y pueden acceder al sistema una gran cantidad de usuarios. No es necesario

que el ordenador sea de grandes prestaciones pues es una aplicación ligera y consume pocos recursos del equipo en el que es ejecutada.

Para el desarrollo del sistema se hace necesario utilizar determinadas herramientas y lenguajes para facilitar el trabajo y la modelación del mismo. A continuación se detallan las características significativas que definieron su uso.

1.5.3 Lenguaje de modelado

UML 2.0

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es un lenguaje visual que permite especificar, construir y documentar los artefactos del sistema de software respondiendo a un enfoque orientado a objetos. Posibilita hacer el diseño más accesible para los miembros del equipo de desarrollo, así como la estandarización del mismo, de modo que se comprenda mejor el negocio (IBM, 2015). Se hizo uso de este para la creación de datos ya que nos permite describir la estructura de datos del sistema, el tipo de datos y la forma en que estos se relacionan.

1.5.4 Herramientas utilizadas para el modelado

Visual Paradigm for UML 8.0

Es una herramienta CASE que soporta el modelado mediante UML. Facilita la reutilización, pues se dispone de una herramienta centralizada donde se encuentran los modelos utilizados en cada proyecto creado. Permite visualizar el flujo central detallado de cada proceso mediante diagramas, posibilitando la obtención de los mismos definidos por la metodología escogida, entre ellos el diagrama de clase, el modelo de datos, entre otros (UML, Visual Paradigm for, 2015). Se selecciona por la fuerte integración que posee con el lenguaje UML. También fue muy influyente ya que unido a las ventajas que brinda, en la UCI se cuenta con la licencia para su uso.

1.5.5 Selección del entorno de desarrollo

IDE NetBeans 8.0

NetBeans 8.0 es un IDE que permite desarrollar aplicaciones de escritorio, móviles y aplicaciones web, así como trabajar con HTML5, JavaScript y CSS, posibilita programar en distintos lenguajes. Ofrece un excelente entorno para programar en PHP. Cuenta con las características de multiplataforma, integración con sistemas de control de versiones, integración con Marcos de Trabajo, depurar y ejecutar programas, importar y exportar proyectos. Es gratuito y de código abierto con una gran comunidad de usuarios y desarrolladores de todo el mundo (NetBeans, 2015). Se decide utilizar este

IDE porque es un producto de código abierto, gratuito y libre, además permite la creación de aplicaciones web y la utilización del marco de trabajo Symfony.

1.5.6 Marco de trabajo

Symfony 2.3.7

"Symfony es un marco de trabajo diseñado para optimizar el desarrollo de las aplicaciones web, aumentando la calidad de trabajo y la productividad. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja" (Potencier, 2008). Es un marco de trabajo PHP basado en la arquitectura Modelo Vista Controlador (MVC, por sus siglas en inglés, Model View Controller). Symfony se distribuye bajo licencia Open Source MIT[1], que no impone restricciones y permite el desarrollo de aplicaciones de código. Cuenta con un amplio soporte para la seguridad del sistema (Symfony, 2003). Es seleccionado teniendo en cuenta que fue diseñado para optimizar el desarrollo de aplicaciones web, proporcionando diferentes tecnologías para agilizar aplicaciones complejas y guiando al desarrollador a acostumbrarse al orden y buenas prácticas dentro del proyecto.

ORM Doctrine 2.3

Doctrine es un marco de trabajo que proporciona persistencia transparente de objetos PHP, el cual se situa en la parte superior de una capa de abstracción de base de datos (DBAL, por sus siglas en inglés, Database Abstraction Layer). Una de sus principales características es que cuenta con la opción de escribir las consultas de base de datos en un lenguaje de consulta estructurado (SQL, por sus siglas en inglés Structured Query Language) orientado a objetos, llamado lenguaje de consulta Doctrine (DQL[2], por sus siglas en inglés, Doctrine Query Language) (Doctrine, 2014). Es una librería completa y configurable además, viene integrada por defecto con Symfony2 y presenta entre sus características principales las siguientes:

1. Generación automática del modelo.
2. Posibilidades de trabajar con YAML.
3. Relaciones entre entidades.

[1] Open Source MIT: Licencia de código abierto que otorga el Instituto Tecnológico de Massachusetts dentro de un proyecto que persigue una ubicación central para almacenar, mantener y dar seguimiento de software de código abierto que se desarrolla dentro de la comunidad del MIT.

[2] DQL: Es un lenguaje creado para ayudar al programador a extraer objetos de la base de datos. Es importante considerar su uso para mejorar el rendimiento.

4. Lenguajes DQL.

Doctrine es seleccionado por la necesidad de utilizar un marco de trabajo que permite trabajar con los objetos de la base de datos. Además viene integrado por defecto al marco de trabajo Symfony2 y permite la creación del modelo a través del esquema de base de datos.

1.5.7 Lenguajes de programación

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente (Programación, 2015).

Del lado del Servidor:

PHP 5.4

PHP (Acrónimo de Hypertext Preprocessor), es un lenguaje de código abierto originalmente diseñado para el desarrollo web. El código es ejecutado en el servidor, generando HTML y enviándolo al cliente. Es un lenguaje multiplataforma con capacidad de conexión con la mayoría de los manejadores de base de datos, entre ellos PostgreSQL, además de su soporte para extensiones de base de datos como la capa de abstracción de base de datos y soporte para comunicarse con otros servicios usando protocolos tales como el protocolo ligero de acceso a directorios (LDAP, por sus siglas en inglés, Lightweight Directory Access Protocol*)*, protocolo para la transferencia simple de correo electrónico (SMTP, por sus siglas en inglés, Simple Mail Transfer Protocol), entre otros. Permite la utilización de las técnicas de programación orientada a objetos (PHP, 2015). Es seleccionado por ser un lenguaje de programación del lado del servidor que posee disímiles características para facilitar el desarrollo de una aplicación web, es fácilmente integrable con el marco de trabajo escogido y permite la programación orientada a objeto.

Motor de Plantillas Twig 2.1

Es un motor de plantillas, que posee un ambiente amigable para el diseñador y el desarrollador, permitiendo añadir funcionalidades a los entornos de plantillas. Posee una sintaxis corta y concisa, similar a la de otros lenguajes de programación. Además implementa un mecanismo de herencia de plantillas, para acelerar el rendimiento del sistema que se desarrolla (Pacheco, 2013).

Este lenguaje permite compilar las plantillas hasta código PHP regular optimizado, lo que proporciona rapidez durante la implementación. Garantiza la seguridad del código de las plantillas ya que posee un modo de recinto de seguridad para evaluar el código de plantilla que no es confiable. Permite al desarrollador definir sus propias etiquetas filtros personalizados, garantizando flexibilidad a la aplicación. Posibilita la depuración de las plantillas creadas, mediante mensajes de error con el nombre del archivo y el nú mero de línea donde se produjo el problema (Pacheco, 2013). Twig es seleccionado por la integración que posee con el marco de trabajo escogido. Además, permite agilizar la construcción de plantillas de la vista, brindándoles estructuración y velocidad de ejecución del lado del cliente. Este lenguaje permite compilar las plantillas hasta código PHP optimizado, lo que proporciona rapidez durante la implementación (Twig, 2012).

Del lado del Cliente:

JavaScript 1.6

Es un lenguaje de programación utilizado para crear programas encargados de realizar acciones dentro del ámbito de una página web. Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Su uso se basa fundamentalmente en la utilización de contenidos dinámicos en las páginas web (Eguiluz, 2010). JavaScript es seleccionado por la necesidad de contar con un lenguaje para validar el contenido del lado del cliente, así como para manejar algunos datos y eventos que son necesarios y requeridos en las funcionalidades a desarrollar en la aplicación.

HTML 5.0

Es un lenguaje de marcado de hipertexto (HTML, por sus siglas en ingles), es decir, documentos de texto estructurados, incluye enlaces que conducen a otros documentos o a otras fuentes de información y permite la inclusión de información por formularios, entre otros. HTML 5 se caracteriza por ofrecer una mejor estructura eliminando el uso excesivo de las etiquetas <*div*>, esta se emplea para definir un bloque de contenido o sección de la página; con el objetivo de que la web sea más coherente y comprensible (HTML 5). Es seleccionado debido a que es un lenguaje estático para el desarrollo de sitios web que permite describir hipertexto presentando el texto de forma estructurada y agradable. Se utiliza para definir texto, tablas y otros elementos que forman parte del diseño de la página. Además incluye script (por ejemplo *JavaScript* y PHP), mejoras en los formularios y brinda facilidades para validar el contenido.

CSS 3.0

La hoja de estilos en cascadas (CSS, por sus siglas en inglés) es un lenguaje creado para controlar la presentación de los documentos electrónicos definidos con HTML. CSS es la mejor forma de separar los contenidos de su presentación y optimiza el trabajo para la creación de páginas web complejas. Es un lenguaje que trabaja junto a HTML para proveer estilos visuales a los elementos del documento como tamaño, color, fondo y borde (Gauchat, 2012). CSS3 es seleccionado por las ventajas y facilidades que aporta a la hora de trabajar las plantillas que se van a mostrar en la aplicación, permitiendo ahorro de código y tiempo a los desarrolladores en la elaboración de varios trucos para lograr confeccionar las interfaces como son requeridas por el cliente (CSS3, 2015).

1.5.8 Herramientas para base de datos

Gestor de Base de Datos PostgreSQL 9.1

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[3] y con su código fuente disponible libremente. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema (PostgreSQL, 2015). Se decidió usar PostgreSQL, ya que es un sistema de gestión de base de datos relacional orientado a objetos, multiplataforma y libre.

1.5.9 Servidor web

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor. Realiza conexiones bidireccionales y/o unidireccionales, síncronas o asíncronas con el cliente, generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente. Para la transmisión de todos estos datos generalmente se utiliza el protocolo HTTP[4] o el protocolo HTTPS[5] (la versión cifrada y autenticada), para estas comunicaciones pertenecientes a la capa de aplicación del modelo OSI[6] (Servidores, 2015).

Apache 2.1

[3]Berkeley Software Distribution o BSD (en español, «distribución de software Berkeley»), licencia de software libre que permite ver el código y modificarlo pero también permite cerrar el sistema o la aplicación.
[5]HyperText Transfer Protocol (en español protocolo de transferencia de hipertexto).
[6]Secure HTTP.
[7]Open System Interconnection (modelo de referencia de Interconexión de Sistemas Abiertos).

Es un servidor web de los más utilizados en el mundo por las ventajas que brinda, es de código abierto, multiplataforma y flexible debido a que es altamente configurable y de diseño modular (Project, 2015). Esta herramienta trabaja con varios lenguajes de script, entre ellos PHP, el cual ha sido seleccionado para la implementación. Presenta abundante documentación posibilitando así un mejor trabajo por parte de los miembros del equipo de desarrollo. Es seleccionado por la necesidad de un servidor web compatible con la plataforma de desarrollo, que permita rapidez en la navegación y la seguridad e integridad de los datos con los que se trabajen. Además Apache permite configurar los módulos a utilizar en el servidor, logrando desechar elementos innecesarios para lograr mayor rapidez en la renderización de las páginas.

1.6 Patrones

1.6.1 Patrón de arquitectura

Un patrón de arquitectura de software es un esquema genérico probado para solucionar un problema particular recurrente que surge en un cierto contexto. Este esquema se especifica describiendo los componentes, con sus responsabilidades, relaciones y las formas en que colaboran (Rabí, y otros, 2010).

Modelo Vista Controlador (MVC)

El Modelo Vista Controlador es un patrón de arquitectura que divide una aplicación interactiva en tres componentes: el modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio. La vista transforma el modelo en una página web que permite al usuario interactuar con ella. El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

El empleo de este patrón proporciona reutilización de los componentes, facilidad para la realización de pruebas funcionales a los mismos y simplicidad en el mantenimiento del sistema.

1.6.2 Patrones de diseño

Los diseñadores expertos no resuelven los problemas desde el principio, reutilizan soluciones que han funcionado en el pasado. Se encuentran patrones de clases y objetos de comunicación recurrentes en muchos sistemas orientados a objetos. Estos patrones resuelven problemas de diseño específicos y hacen el diseño flexible y reusable. Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptado para resolver un problema de diseño general en un contexto particular (Addison Wesley Helm, y otros, 1995).

La utilización de patrones de diseño, permite ahorrar grandes cantidades de tiempo en la construcción de software. El producto obtenido es más fácil de comprender, mantener y extender. Existen versiones ya implementadas de estas funcionalidades comunes (marcos de trabajo) que permiten centrarse en desarrollar solo la funcionalidad específica requerida por cada aplicación y que además, dan mejor imagen de profesionalidad y calidad. Los patrones de diseño se agrupan en dos grandes categorías: GRASP[7] y GOF[8]. Los primeros describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Dentro de este grupo de patrones se encuentran los siguientes: Experto, Creador, Bajo Acoplamiento, Controlador y Alta Cohesión. Los patrones de diseño GOF son 23 y se clasifican según su propósito en Creacionales, Estructurales, Comportamiento y según su ámbito en Objeto y de Clase.

1.7 Métricas

Las métricas de software tienen un papel decisivo en la obtención de un producto de alta calidad, porque determinan mediante estadísticas basadas en la experiencia, el avance de software y el cumplimiento de parámetros requeridos (Rabí, y otros, 2010).

1.7.1 Métricas para requisitos

Estabilidad de los requisitos

Es necesario lograr una estabilidad en los requisitos para el correcto funcionamiento de los demás flujos de trabajo. El objetivo de esta métrica es medir la estabilidad de los requisitos para asegurar su adecuación antes de pasar al próximo flujo de trabajo. Se considera que los requisitos son estables cuando no existen adiciones o supresiones en ellos que impliquen modificaciones en las funcionalidades principales de la aplicación (requisitos, 2015).

$$ETR = \left[\frac{RT - RM}{RT}\right] \times 100$$

Donde:

1. ETR: valor de la estabilidad de los requisitos.
2. RT: total de requisitos definidos.

[7] GRASP: Acrónimo de "General Responsibility Assignment Software Patterns" en español Patrones generales de software para asignación de responsabilidades.

[8] Gang of Four, en español Banda de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

3. RM: número de requisitos modificados, que se obtienen como la sumatoria de los requisitos insertados, modificados y eliminados.

Esta métrica ofrece valores entre 0 y 100. El mejor valor de ETR es el más cercano a 100 porque mostrará que no se están realizando cambios sobre los requisitos, son estables y por tanto es confiable trabajar el análisis y diseño sobre ellos.

Especificidad de los requisitos

El objetivo de esta métrica es cuantificar la especificidad o falta de ambigüedad en la definición de los requisitos. Para calcular esta métrica deben contarse los requisitos que tuvieron igual interpretación por los revisores y compararlos con el total de requisitos definidos. El valor de esta métrica debe estar siempre entre 0 y 1. Mientras más cerca de 1 esté el valor de la especificidad de los requisitos mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad en la especificación de los requisitos (Pressman, 2000). La misma se calcula como:

$$ER = {}^{n_{ui}}/_{n_r}$$

Donde:

- ✓ ER: grado de especificidad de los requisitos.
- ✓ nui: número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas.
- ✓ nr: cantidad de requisitos en especificación.

El valor de esta métrica debe estar siempre entre 0 y 1. Mientras más cerca de 1 esté el valor de ER mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad en la especificación de los requisitos.

Grado de validación de los requisitos

Los requisitos deben ser posibles de validar. La validación de los requisitos se realiza en consenso del equipo de desarrollo al contrastar lo que desea el cliente con la posibilidad real de implementarlo. El grado de validación de los requisitos mide la corrección en la definición de los requisitos (Métricas, 2015).

$$VR = {}^{n_c}/_{(n_c + n_{nv})}$$

Donde:

- ✓ VR: grado de validación de los requisitos.

- ✓ nc: número de requisitos que se han validado como correctos.
- ✓ nnv: número de requisitos no validados aún.

El resultado de esta métrica está siempre entre 0 y 1. El valor óptimo de esta métrica es el más cercano a 1 e indica un alto nivel de corrección en la definición de los requisitos.

1.7.2 Métricas para el diseño

Las métricas referentes al tamaño para las clases orientadas a objetos (OO) se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema OO como un todo. El tamaño general de una clase puede medirse determinando las siguientes medidas:

- ✓ El total de operaciones (heredadas y privadas de la instancia), que se encapsulan dentro de la clase.
- ✓ El número de atributos (heredados y privados de la instancia), encapsulados por la clase.

Estos dos valores son sumados de acuerdo a la clase que se analiza y los resultados son tomados como cantidad de procedimientos (CP) que luego son comparados para determinar el TOC de cada clase. (Sánchez Fornaris, 2009)

Clasificación	Valores
Pequeño	CP<=20
Mediano	CP>20 y CP<=30
Grande	CP>30

Tamaño Operacional de Clase (TOC)

Consiste en medir el tamaño general de una clase tomando el valor de la cantidad de operaciones que están encapsuladas dentro de dicha clase (Lincke, y otros, 2008).

Si el resultado obtenido indica valores grandes, significa que la clase posee un alto grado de responsabilidad, debido a esto se reducirá la reutilización, se hará mucho más difícil la implementación y la realización de pruebas de dicha clase. Por tanto mientras menor sea el valor para el TOC se hará mucho más fácil la reutilización de dicha clase dentro del sistema.

Atributo	Categoría	Criterio

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

Tabla 1 Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica

Relaciones entre Clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad (Web, 2015):

Acoplamiento: un aumento del RC implica un aumento del acoplamiento de la clase.

Complejidad de mantenimiento: un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.

Cantidad de pruebas: un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

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

	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio

Tabla 2 Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica

1.8 Pruebas

La (IEEE, 2015) define las pruebas de software como una actividad en la que un sistema o un componente es ejecutado bajo condiciones especificadas. El objetivo es diseñar una serie de casos de pruebas que tengan una alta probabilidad de encontrar errores. Para ello se aplican las técnicas de pruebas del software, las cuales facilitan una guía sistemática para diseñar pruebas que comprueben la lógica interna de los componentes de software y verifiquen los dominios de entrada y salida del programa para descubrir errores en la funcionalidad, el comportamiento y rendimiento (Pressman, 2000).

1.8.1 Método de prueba de Caja Negra

Las pruebas de caja negra son aquellas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del funcionamiento del sistema sin tener mucho en cuenta la estructura interna del software (Peña, 2010). Estas pruebas permiten encontrar:

1. Funciones incorrectas o ausentes.

2. Errores de interfaz.

3. Errores en estructuras de datos o en accesos a las Bases de Datos externas.

4. Errores de rendimiento.

5. Errores de inicialización y terminación.

Según (Pressman, 2000) para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

1. Técnica de la Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

2. Técnica del Análisis de Valores Límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
3. Técnica de Grafos de Causa-Efecto: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

En este caso se escoge dentro del método de Caja Negra la técnica de la Partición de Equivalencia que es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así en número de casos de pruebas que hay que desarrollar.

1.8.2 Método de prueba de Caja Blanca

El método de prueba de caja blanca, denominado a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero de software puede obtener casos de prueba que: (1) garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo; (2) ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; (3) ejecuten todos los bucles en sus límites y con sus límites operacionales; y (4) ejerciten las estructuras internas de datos para asegurar su validez (Pressman, 2002).

Para ejecutar este tipo de pruebas según (Pressman, 2002) se utilizará la técnica del camino básico: esta técnica permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

1.8.3 Pruebas de aceptación

La prueba de aceptación es generalmente desarrollada y ejecutada por el cliente o un especialista de la aplicación y es conducida a determinar cómo el sistema satisface sus criterios de aceptación validando los requisitos que han sido levantados para el desarrollo, incluyendo la documentación y procesos de negocio. Está considerada como la fase final del proceso para crear una confianza en que el producto es el apropiado para su uso en explotación (Pressman, 2000).

1.9 Conclusiones parciales

Al finalizar el presente capítulo se concluye que:

- ✓ El estudio de los diferentes sistemas de gestión de la calidad de software permitió demostrar que las soluciones analizadas no se ajustan a las necesidades del Grupo de Calidad del centro CEGEL.
- ✓ La metodología, las herramientas y lenguajes de programación seleccionados son los adecuados para el desarrollo del sistema, además responden a la soberanía tecnológica estableciendo las bases para el desarrollo de la propuesta de solución.
- ✓ Las métricas y los tipos de pruebas seleccionados permiten identificar deficiencias en los requisitos, en el diseño y en la implementación de las funcionalidades del sistema, con el objetivo de solventarlas y obtener un sistema que contribuya a solucionar el problema planteado.

CAPÍTULO 2: PLANIFICACIÓN, DISEÑO E IMPLEMENTACIÓN DE LA PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se hace una descripción de la solución propuesta, teniendo en cuenta las fases de Planificación, Diseño y Codificación del ciclo de vida de la metodología XP. Se identifican los requisitos funcionales y no funcionales que debe cumplir la solución a desarrollar, posteriormente se confeccionan las historias de usuario y demás artefactos definidos por la metodología. Se identifican y organizan las clases relevantes para las funcionalidades del sistema, así como los patrones arquitectónicos y de diseño utilizados para la realización de la aplicación.

2.2 Propuesta de solución

Teniendo en cuenta las dificultades que presentan cada una de las herramientas utilizadas por el grupo de calidad, mencionadas en la fundamentación teórica, se decide desarrollar un sistema que integre dichas herramientas de manera que contribuya a la mejora del proceso de gestión de la calidad.

Mediante la utilización del sistema el Asesor de la calidad puede realizar la planificación anual de revisiones, asignar tareas a los revisores, realizar minutas de reunión de apertura y cierre de los distintos proyectos del centro, enviando notificaciones a los involucrados por vía correo electrónico. Además realiza todas las actividades del Revisor líder y del Revisor. Gestiona las listas de chequeo para poder detectar no conformidades en las revisiones que se les realizan a los sistemas, para posteriormente darle seguimiento a cada una de las no conformidades detectadas por los revisores. También gestiona la asignación para evaluar las características de calidad de los proyectos del centro y así lograr una mejor claridad y precisión en el trabajo realizado. Por otra parte el Evaluador puede evaluar el grado de implementación de las características de calidad por características y por disciplina permitiéndole obtener el avance en cuanto a implementación de los proyectos del centro, mientras que el Jefe de proyecto podrá darle seguimiento a cada uno de sus proyectos para chequear que los resultados sean los más positivos posibles.

El sistema servirá de apoyo en el proceso de gestión de la calidad como un todo, contribuirá a la evaluación de la satisfacción de los clientes internos en el proceso de desarrollo de software, a la revisión de los procesos y productos de trabajo que se llevan

a cabo en los proyectos del centro CEGEL y al grado de implementación de las características de calidad. Además el sistema será capaz de gestionar cada uno de los procesos que se llevan a cabo en el Grupo de Calidad del Centro de Gobierno Electrónico.

2.3 Requisitos

2.3.1 Técnicas para la captura de requisitos

El levantamiento de requisitos es una etapa esencial en el arranque de todo proyecto de desarrollo de software y debe de realizarse efectivamente para elevar las probabilidades de éxito de los proyectos. Soluciona las posibles diferencias entre las personas involucradas en el mismo, con el propósito de definir y refinar los requisitos para cumplir las restricciones acordadas por ambas partes. El proceso de levantamiento de requisitos soporta el desarrollo de la especificación de los requisitos, de tal forma que tengan los siguientes atributos: ser completos, consistentes y han de estar dentro del alcance del proyecto, tener un único identificador, cumplir con los objetivos de los clientes, ser viables y apropiados para el desarrollo, además de tener capacidad de prueba (Torres, 2012). Para ello fueron utilizadas las siguientes técnicas:

Entrevista: se emplea para la obtención de requisitos por ser la mejor fuente de información cualitativa. Tiene como ventaja ser una oportunidad para que el analista conozca el grado de aceptación o resistencia que existe entre los usuarios hacia el sistema que se desea diseñar.

Tormenta de ideas: es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Puede ayudar a generar una gran variedad de vistas del problema y a formularlo de diferentes formas, sobre todo al comienzo del proceso de captura, cuando los requisitos son todavía muy difusos. Sin embargo, también se requiere participación intensiva del analista (Zapata, y otros, 2007).

A partir de los resultados de aplicar las técnicas de obtención de requisitos se obtuvieron los siguientes requisitos funcionales:

RF_1 Autenticar usuarios.

RF_2 Adicionar proyecto

RF_3 Editar proyecto

RF_4 Listar proyectos

RF_5 Mostrar proyecto

RF_6 Eliminar proyecto

RF_7 Adicionar módulo

RF_8 Editar módulo

RF_9 Listar módulos

RF_10 Mostrar módulo

RF_11 Eliminar módulo

RF_12 Adicionar dimensión

RF_13 Editar dimensión

RF_14 Listar dimensiones

RF_15 Mostrar dimensión

RF_16 Eliminar dimensión

RF_17 Adicionar datos de nomencladores

RF_18 Editar datos de nomencladores

RF_19 Listar datos de nomencladores

RF_20 Mostrar datos de nomencladores

RF_21 Eliminar datos de nomencladores

RF_22 Adicionar usuario

RF_23 Editar usuario

RF_24 Listar usuarios

RF_25 Mostrar usuario

RF_26 Eliminar usuario

RF_27 Adicionar rol

RF_28 Editar rol

RF_29 Listar roles

RF_30 Mostrar rol

RF_31 Eliminar rol

RF_32 Adicionar pregunta de evaluación del servicio brindado

RF_33 Editar pregunta de evaluación del servicio brindado

RF_34 Listar preguntas de evaluaciones del servicio brindado

RF_35 Mostrar pregunta de evaluación del servicio brindado

RF_36 Eliminar pregunta de evaluación del servicio brindado

RF_37 Establecer permisos de usuario

RF_38 Adicionar planificación anual de revisiones

RF_39 Editar planificación anual de revisiones

RF_40 Listar planificación anual de revisiones

RF_41 Mostrar planificación anual de revisiones

RF_42 Eliminar planificación anual de revisiones

RF_43 Adicionar tarea

RF_44 Editar tarea

RF_45 Listar tareas

RF_46 Mostrar tarea

RF_47 Eliminar tarea

RF_48 Adicionar revisión de proyecto

RF_49 Editar revisión de proyecto

RF_50 Listar revisiones de proyectos

RF_51 Mostrar revisión de proyecto

RF_52 Eliminar revisión de proyecto

RF_53 Adicionar notificación de revisión

RF_54 Editar notificación de revisión

RF_55 Listar notificaciones de revisión

RF_56 Mostrar notificación de revisión

RF_57 Eliminar notificación de revisión

RF_58 Enviar notificación de revisión

RF_59 Adicionar minuta de reunión

RF_60 Editar minuta de reunión

RF_61 Listar minutas de reunión

RF_62 Mostrar minuta de reunión

RF_63 Eliminar minuta de reunión

RF_64 Adicionar elementos de la lista de chequeo

RF_65 Editar elementos de la lista de chequeo

RF_66 Listar elementos de la lista de chequeo

RF_67 Mostrar elementos de la lista de chequeo

RF_68 Eliminar elementos de la lista de chequeo

RF_69 Adicionar no conformidad

RF_70 Editar no conformidad

RF_71 Listar no conformidades

RF_72 Mostrar no conformidad

RF_73 Eliminar no conformidad

RF_74 Adicionar seguimiento de no conformidad

RF_75 Editar seguimiento de no conformidad

RF_76 Listar seguimiento de no conformidades

RF_77 Mostrar seguimiento de no conformidad

RF_78 Eliminar seguimiento de no conformidad

RF_79 Adicionar evaluación de proyecto

RF_80 Editar evaluación de proyecto

RF_81 Listar evaluaciones de proyectos

RF_82 Mostrar evaluación de proyecto

RF_83 Eliminar evaluación de proyecto

RF_84 Visualizar reporte de indicador grado de implementación de los procesos

RF_85 Visualizar reporte de indicador adherencia a criterios de evaluación.

RF_86 Visualizar reporte de indicador distribución de impacto y tipo de no conformidades

RF_87 Adicionar guía de calidad

RF_88 Editar guía de calidad

RF_89 Listar guías de calidad

RF_90 Mostrar guía de calidad

RF_91 Eliminar guía de calidad

RF_92 Insertar característica

RF_93 Editar característica

RF_94 Listar características

RF_95 Mostrar característica

RF_96 Eliminar característica

RF_97 Insertar disciplina

RF_98 Editar disciplina

RF_99 Listar disciplinas

RF_100 Mostrar disciplina

RF_101 Eliminar disciplina

RF_102 Insertar pregunta

RF_103 Editar pregunta

RF_104 Listar preguntas

RF_105 Mostrar pregunta

RF_106 Eliminar pregunta

RF_107 Insertar asignación de evaluación

RF_108 Listar asignaciones de evaluaciones

RF_109 Mostrar asignación de evaluación

RF_110 Eliminar asignación de evaluación

RF_111 Evaluar el grado de implementación

RF_112 Visualizar reporte del grado de implementación general del centro

RF_113 Visualizar reporte del grado de implementación por proyectos

RF_114 Visualizar reporte del grado de implementación por características en el centro

RF_115 Visualizar reporte del grado de implementación por características en los proyectos

RF_116 Visualizar reporte de las preguntas evaluadas negativamente

RF_117 Visualizar reporte del grado de implementación general por módulo

RF_118 Visualizar reporte del grado de implementación del módulo por características

RF_119 Guardar reporte del grado de implementación por proyectos

RF_120 Guardar reporte del grado de implementación por características en el centro

RF_121 Guardar reporte del grado de implementación por características en los proyectos

RF_122 Guardar reporte de las preguntas evaluadas negativamente

RF_123 Guardar reporte del grado de implementación del módulo por características

RF_124 Adicionar encuesta

RF_125 Adicionar grupo de preguntas

RF_126 Adicionar pregunta

RF_127 Modificar pregunta

RF_128 Modificar Grupo de preguntas

RF_129 Eliminar pregunta

RF_130 Listar encuestas

RF_131 Eliminar encuesta

RF_132 Mostrar índice de satisfacción del cliente interno a nivel de rol.

RF_133 Mostrar índice de satisfacción de cada dimensión a nivel de rol.

RF_134 Mostrar índice de satisfacción de cada elemento de una dimensión a nivel de rol

2.3.2 Requisitos no Funcionales

Los requisitos no funcionales, son restricciones de los servicios ofrecidos por el sistema. De forma alternativa, se utilizan para definir las restricciones de la aplicación como la capacidad de los dispositivos de entrada/salida y restricciones de tiempo sobre el sistema (Requerimientos, 2015). La metodología XP no incluye la descripción de estos requisitos en las Historias de Usuario, debido a que los clientes generalmente no están familiarizados con las terminologías técnicas que se utilizan para describir las mismas, siendo el equipo de desarrollo el encargado de capturar estos requisitos a partir del intercambio de información con el cliente. A continuación se presentan los requisitos no funcionales identificados:

Disponibilidad

RNF_1 El sistema debe estar disponible todo el tiempo para sus usuarios, descontando el tiempo en que se encuentre en mantenimiento.

RNF_2 El período entre fallos recuperables, como por ejemplo fallos en el servidor principal no debe exceder las 24 horas.

Usabilidad

RNF_3 El sistema podrá ser usado por personas con conocimientos básicos en el manejo de computadoras.

RNF_4 Los usuarios deberán tener conocimiento de la forma en que se realizan los procesos que maneja el sistema.

Eficiencia

RNF_5 Tiempo de respuesta: el promedio de las peticiones que se realizan al servidor no deben ser mayor de 3 segundos. En el caso de información que involucre consultas a las bases de datos los tiempos de respuestas no deben exceder los 10 segundos.

Soporte

RNF_6 La aplicación debe estar documentada y proveer el código fuente, previendo futuras modificaciones en el mismo para potenciar su alcance o eficiencia.

RNF_7 Consta con la documentación necesaria para el aprendizaje sobre el uso de la herramienta informática.

Interfaz

RNF_8 Debe mostrar una interfaz gráfica agradable y sencilla al usuario capaz de guiar la navegación del mismo a través del sistema.

RNF_9 Se usarán colores que den una combinación agradable a la aplicación.

Seguridad

RNF_10 La seguridad está a nivel de gestión de roles con el fin de mantener la integridad de los datos, por el cual el acceso a la herramienta será a través de estos roles, trayendo consigo además la protección de la información.

Portabilidad

RNF_11 El sistema debe ser multiplataforma con enfoque en tecnologías basadas en software no propietario.

Requisitos de hardware requerido para utilizar la aplicación web

RNF_12 La comunicación entre el servidor de aplicaciones y la base de datos se lleva a través del protocolo TCP/IP.

RNF_13 La comunicación entre el cliente y el servidor de aplicaciones se lleva a través del protocolo HTTP.

- ✓ **Cliente**

Deben cumplir con los siguientes requisitos de hardware:

RNF_14 Ordenador Pentium IV con 1.7 GHz de velocidad de microprocesador o superior.

RNF_15 Memoria RAM mínimo 256MB.

- ✓ **Servidor**

Debe cumplir con los siguientes requisitos de hardware:

RNF_16 Ordenador Pentium IV con 1.7 GHz de velocidad de microprocesador o superior.

RNF_17 Memoria RAM mínimo 512MB.

Requisitos de software

✓ **Cliente**

RNF_18 Navegador web Mozilla Firefox 13.0 o superior.

✓ **Servidor**

RNF_19 Servidor Web Apache 2.2 o superior, con módulo PHP 5.3 o superior.

RNF_20 Gestor de base de datos PostgreSQL 9.1 o superior (para la pc servidor).

2.3.3 Actores del sistema

Se define como actores del sistema a todas las personas del Grupo de Calidad del Centro de Gobierno Electrónico que están involucradas en el proceso de gestión de la calidad como un todo.

Asesor de la calidad: realiza la planificación anual de las revisiones, asigna tareas a los revisores líderes y se encarga del análisis de tendencias. Puede realizar todas las actividades del Revisor líder y del Revisor. Además consulta los resultados de la evaluación del servicio brindado, así como gestionar las preguntas para el mismo. Gestiona proyectos, módulos, usuarios, elementos de la lista de chequeo y los datos de los nomencladores.

Revisor líder: responsable en la aplicación del proceso total de revisión de calidad a proceso y producto. Controla además, la generación de los artefactos del proceso de revisión y terminado el proceso redacta el informe final. Le da seguimiento a la revisión realizada. Puede realizar todas las actividades del Revisor. Además envía notificación de revisión y gestiona la misma.

Revisor: responde la lista de chequeo para generar las no conformidades en caso que existan. Participa en la reunión de inicio y cierre de la evaluación.

Jefe de proyecto: actualiza el registro de evaluaciones del proyecto con el monitoreo del estado de las no conformidades y sus acciones correctivas. Evalúa el servicio brindado por el Grupo de Calidad.

Administrador: actualiza el registro de evaluaciones del proyecto con el monitoreo del estado de las no conformidades, sus acciones correctivas y se encarga de establecer los permisos a los usuarios del sistema.

2.4 Planificación

El ciclo de vida de un proyecto realizado con la metodología XP inicia con la fase de planificación. En esta fase, los clientes plantean a grandes rasgos las Historias de Usuario (HU). Al finalizar el equipo cuenta con suficiente material de trabajo como para producir una primera entrega. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas y tecnologías que serán utilizadas en el proyecto. El objetivo de esta fase es llegar a un acuerdo entre los clientes y los programadores respecto a cuáles serán las HU a ser implementadas durante cada iteración y establecer cuál va a ser el contenido de la primera entrega. Los programadores estiman cuánto tiempo y esfuerzo requiere cada HU y se establece el cronograma (Calabria, y otros, 2003).

Uno de los artefactos que se generan por la metodología de desarrollo XP para la especificación de requisitos del software y las características del sistema son las (HU).

2.4.1 Historias de Usuario

La HU es la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas en las cuales el cliente describe brevemente las características que el sistema debe poseer. El tratamiento de las HU es dinámico y flexible. Cada HU es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. Estas deben proporcionar solo el detalle suficiente como para poder hacer razonable la estimación de cuánto tiempo requiere la implementación de la historia. También se les asigna un número identificativo, una prioridad en el negocio (alta, media, baja) y la iteración en la que se implementará. (Beck, 1999). Para una mejor comprensión de las HU referirse al anexo1.

A continuación se describen las HU de prioridad Alta diseñadas para el desarrollo del sistema.

Historia de Usuario	
Número:1	**Nombre Historia de Usuario:** Gestionar Roles
Modificación de Historia de Usuario Número: ninguna	

Usuario: Administrador/Asesor de la calidad	**Iteración asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 días
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 días
Descripción: • Permite Adicionar Rol El sistema debe mostrar la opción de adicionar rol, donde debe llenar los campos nombre del rol y descripción del rol, luego debe seleccionar el botón Adicionar para guardar los datos. • Permite Modificar Rol El sistema debe permitir la opción de modificar el rol, donde deben aparecer los mismos campos que cuando adicionó el rol, con la posibilidad de rectificar los errores en los indicadores. • Permite Eliminar Rol El sistema debe permitir eliminar un rol, donde seleccionando la opción Eliminar, el sistema muestra un cartel de confirmación de la petición, si selecciona el botón Aceptar se eliminará el rol. • Permite Mostrar Rol El sistema debe ser capaz de mostrar el rol de cada usuario, donde debe aparecer la descripción de cada rol. • Permite Listar Rol El sistema debe ser capaz de mostrar un listado de cada uno de los roles presentes en el sistema.	
Observaciones: solo está autorizado a gestionar los roles, el Administrador del sistema y el Asesor de la calidad.	

Tabla 3 Historia de Usuario Gestionar roles

Historia de Usuario	
Número: 8	**Nombre Historia de Usuario:** Gestionar tarea
Modificación de Historia de Usuario Número: ninguna	
Usuario: Asesor de la calidad/ Revisor líder	**Iteración asignada:** 1
Prioridad en Negocio: Media	**Puntos Estimados:** 2 días

Riesgo en Desarrollo: Media	**Puntos Reales:** 2 días
Descripción: • Permite Adicionar Tarea El sistema debe mostrar la opción adicionar tarea, donde debe llenar los campos Asunto, Descripción, Estado, Prioridad, Asignado a, Fecha de Inicio, Fecha de Cumplimiento, Tiempo Estimado, Porciento Realizado, Iteración Prevista, Evaluación y Complejidad, luego debe seleccionar el botón Adicionar para guardar los datos. • Permite Modificar Tarea El sistema debe permitir la opción de modificar tarea, donde deben aparecer los mismos campos que cuando se adicionó la tarea, con la posibilidad de rectificar los errores en las tareas. • Permite Eliminar Tarea El sistema debe permitir eliminar la tarea, donde seleccionando la opción Eliminar, el sistema muestra un cartel de confirmación de la petición, si selecciona el botón Aceptar se eliminará la tarea. • Permite Mostrar Tarea El sistema debe ser capaz de mostrar las tareas, donde debe aparecer la descripción de cada una de las tareas. • Permite Listar Tarea El sistema debe ser capaz de mostrar un listado de las tareas realizadas en el sistema.	
Observaciones: solo está autorizado a gestionar las tareas, el Asesor de la calidad y el Revisor líder.	

Tabla 4 Historia de Usuario Gestionar tarea

Historia de Usuario	
Número: 5	**Nombre Historia de Usuario:** Gestionar Nomencladores
Modificación de Historia de Usuario Número: ninguna	
Usuario: Administrador/Asesor de la calidad	**Iteración asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 días
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 días

Descripción: • Permite Adicionar Nomenclador El sistema debe mostrar la opción adicionar nomencladores, donde debe llenar el campo Nombre, luego debe seleccionar el botón Adicionar para guardar el dato. • Permite Modificar Nomenclador El sistema debe permitir la opción de modificar nomencladores, donde debe aparecer el mismo campo que cuando adicionó el nomenclador, con la posibilidad de rectificar los errores en el nomenclador. • Permite Eliminar Nomenclador El sistema debe permitir eliminar nomencladores, donde seleccionando la opción Eliminar, el sistema muestra un cartel de confirmación de la petición, si selecciona el botón Aceptar se eliminará el nomenclador. • Permite Mostrar Nomenclador El sistema debe ser capaz de mostrar los nomencladores, donde debe aparecer la descripción de cada uno de los nomencladores. • Permite Listar Nomenclador El sistema debe ser capaz de mostrar un listado de cada uno de los nomencladores.
Observaciones: solo está autorizado a gestionar los nomencladores, el Administrador del sistema y el Asesor de la calidad.

Tabla 5 Historia de Usuario Gestionar Nomencladores

En esta fase el cliente establece la prioridad que tendrá cada HU según sus necesidades más inmediatas, luego los programadores realizan una estimación del esfuerzo que se necesita para cada una de ellas. La planificación es una fase corta, en la que el cliente, y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las HU y, asociadas a estas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas (release[9]). El cronograma fijado en la etapa de planeación se realiza en un número de iteraciones, cada una de ellas tarda de una a cuatro semanas de ejecución.

La planificación se realiza basándose en el tiempo o el alcance. Al planificar por tiempo, se multiplicó el número de iteraciones por el avance del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según el alcance, se dividió la suma

[9] **release: es un ciclo desde la entrevista con el usuario hasta la obtención de una solución.**

de puntos de las Historias de Usuario seleccionadas entre el avance del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

2.4.2 Desarrollo de iteraciones

Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta etapa, generando al final de cada una, un entregable funcional que implementa las HU asignadas a la iteración. Como las HU no tienen suficiente detalle como para permitir su análisis y desarrollo, al principio de cada iteración se realizan las tareas necesarias de análisis, logrando con el cliente todos los datos que sean necesarios. El cliente, por lo tanto, también debe participar activamente durante esta fase del ciclo. Las iteraciones son también utilizadas para medir el progreso del proyecto.

Una vez definidas las HU y estimado el esfuerzo propuesto para la realización de cada una de ellas, se distribuyó la realización del sistema en tres iteraciones, las cuales se describen a continuación de manera más detallada:

Iteración I: en esta iteración se llevará a cabo el desarrollo de las HU del número 1 hasta el número 47.

Iteración II: .en esta iteración se llevará a cabo el desarrollo de las HU del número 48 hasta el número 86.

Iteración III: en esta iteración se llevará a cabo el desarrollo de las HU del número 87 hasta el número 134.

Después de realizada la estimación del esfuerzo y el plan de iteraciones y continuando los pasos que propone XP, se crea el plan de duración de las iteraciones. Este tiene como objetivo fundamental mostrar la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una según la prioridad asignada por el cliente.

Iteración	Historias de Usuario	Duración total de las Iteraciones (semanas)
Iteración I	HU(1-47)	3
Iteración II	HU(47-86)	3
Iteración III	HU(86-134)	5
Total	134	11

Tabla 6 Plan de duración de las iteraciones

A continuación se presenta el plan de entregas definido para la fase de implementación. Atendiendo al mismo se harán entregas del sistema al finalizar cada iteración en la fecha aproximada que se indica en la siguiente tabla.

Iteración	Fecha de Entrega
Iteración I	30 de marzo de 2015
Iteración II	20 de abril de 2015
Iteración III	25 de mayo de 2015

Tabla 7 Distribución del plan de iteraciones

2.5 Diseño de la solución

La metodología XP sugiere que hay que conseguir diseños simples y sencillos. Hay que realizarlo lo menos complicado posible para conseguir un diseño entendible y de fácil implementación, que a la larga costará menos tiempo y esfuerzo desarrollar. Como parte de esta fase, se define la arquitectura del sistema y se precisan los patrones de diseño que se van a emplear para remediar problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño.

2.5.1 Arquitectura del sistema

El sistema se implementa sobre una arquitectura en capas la cual proporciona una organización jerárquica, donde cada capa proporciona servicios a la capa inmediata superior y se sirve de las prestaciones que le brinda la inmediata inferior. La arquitectura está compuesta por: la capa de presentación, la capa de acceso a datos y la capa de datos. Es importante destacar el uso del patrón Modelo Vista Controlador (MVC) en las capas de presentación y acceso a datos. En la figura se puede apreciar la estructura arquitectónica en capas y la ubicación de los componentes en cada una de ellas, destacando los elementos del patrón utilizado. La arquitectura propuesta posee la característica de tener varios complementos transversales a las capas para garantizar seguridad, tratamiento de excepciones, entre otros aspectos. Cada uno de los módulos o bundles[10] se estructuran arquitectónicamente igual. Las entidades del dominio son accesibles en la sub-capa servidor de la capa de presentación y la capa de acceso a datos.

[10] Un bundle es un conjunto estructurado de archivos que implementan una característica única y que puede ser fácilmente compartido con otros desarrolladores.

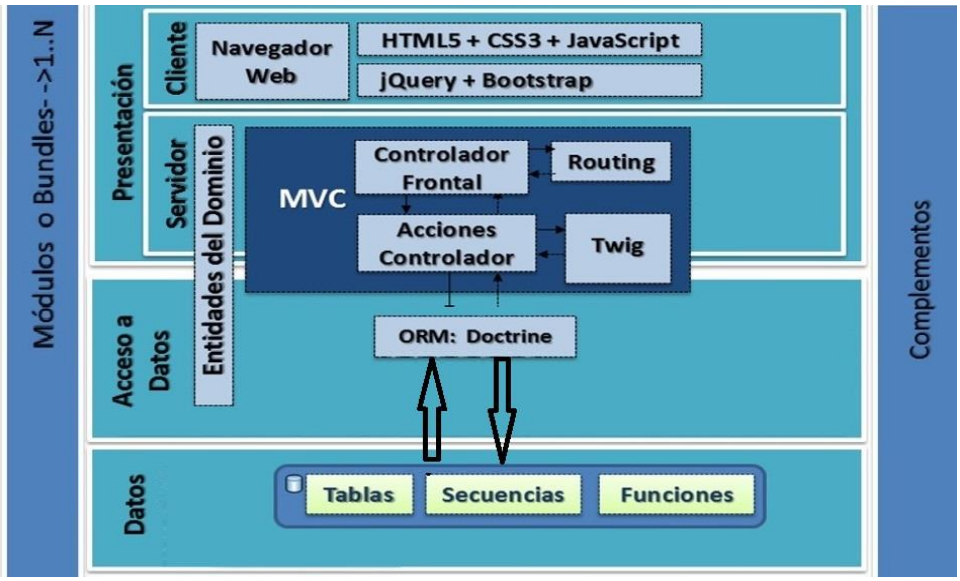

Fig. 1 Esquema de la arquitectura

Capa de Presentación: es la capa que contiene los componentes con los que va a interactuar el usuario (páginas). Permite alcanzar las funcionalidades que brinda el controlador y mostrar o capturar la información a través de los diferentes elementos que comprende: twigs y formularios. Se divide en dos sub-capas: cliente y servidor. En la primera se visualiza mediante el navegador web (utilizando tecnologías como HTML5, CSS3, JavaScript, Bootstrap y JQuery) datos procesados. En la segunda se maneja la lógica de control así como la construcción de páginas y formularios.

Capa de Acceso a datos: contiene las entidades y repositorios que mapea Doctrine como marco de trabajo para la comunicación con el servidor de datos. Gestiona las peticiones de la capa de negocio consultando la base de datos y retorna los datos correspondientes.

Capa de datos: en esta capa se encuentra el gestor de base de datos PostgreSQL y en él un conjunto de esquemas y tablas que permiten persistir la información con la que trabaja la aplicación y manejar su almacenamiento.

Módulos (bundles): cada módulo constituye una estructura de carpetas que se organizan según la arquitectura que se describe en la figura 1. Permiten utilizar funcionalidades construidas por terceros o empaquetar sus propias funcionalidades para distribuirlas y reutilizarlas.

Complementos: los Complementos son un grupo de facilidades y mejoras que permiten lograr una arquitectura más flexible y adaptable tales como gestión de seguridad, de

validaciones, de mensajería y de configuración así como tratamiento de excepciones y otras.

En las capas de presentación y acceso a datos se evidencia el uso del patrón Modelo Vista Controlador (MVC) que se explica a continuación:

Este patrón de arquitectura de software se encarga de separar los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes e incrementa la reutilización y flexibilidad. Divide una aplicación interactiva en tres partes: el modelo contiene los datos y la funcionalidad esencial, las vistas despliegan la información al usuario y los controladores manejan las entradas.

Vista: encapsula las interfaces de usuario, entidades de presentación, plantillas twig y formularios necesarios para la interacción con el cliente.

Controlador: se encarga de recibir una petición, procesar la información, hacer un pedido al modelo y devolver una respuesta a los controladores los cuales a su vez la envían a la vista. Contiene las clases Controller (Controladoras) que se encarga de dar respuesta a las peticiones realizadas por el usuario.

Modelo: contiene las clases Entity (Entidad) y las clases Repository (Repositorio), quienes se encargan del manejo de los datos para visualizarlos.

Inicialmente el usuario interactúa con la interfaz (Vista). El controlador recibe la petición de la acción solicitada y gestiona el evento. El controlador accede al modelo actualizándolo o buscando la información requerida. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario y mostrar los datos del modelo para generar la interfaz apropiada Ver Fig 2.

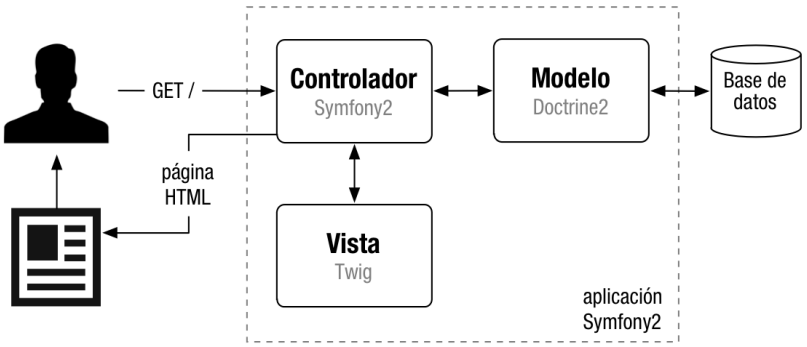

Fig. 2 Patrón MVC

A continuación se muestra la estructura de carpetas del proyecto siguiendo el patrón arquitectónico MVC obtenido con la ayuda del marco de trabajo Symfony2.

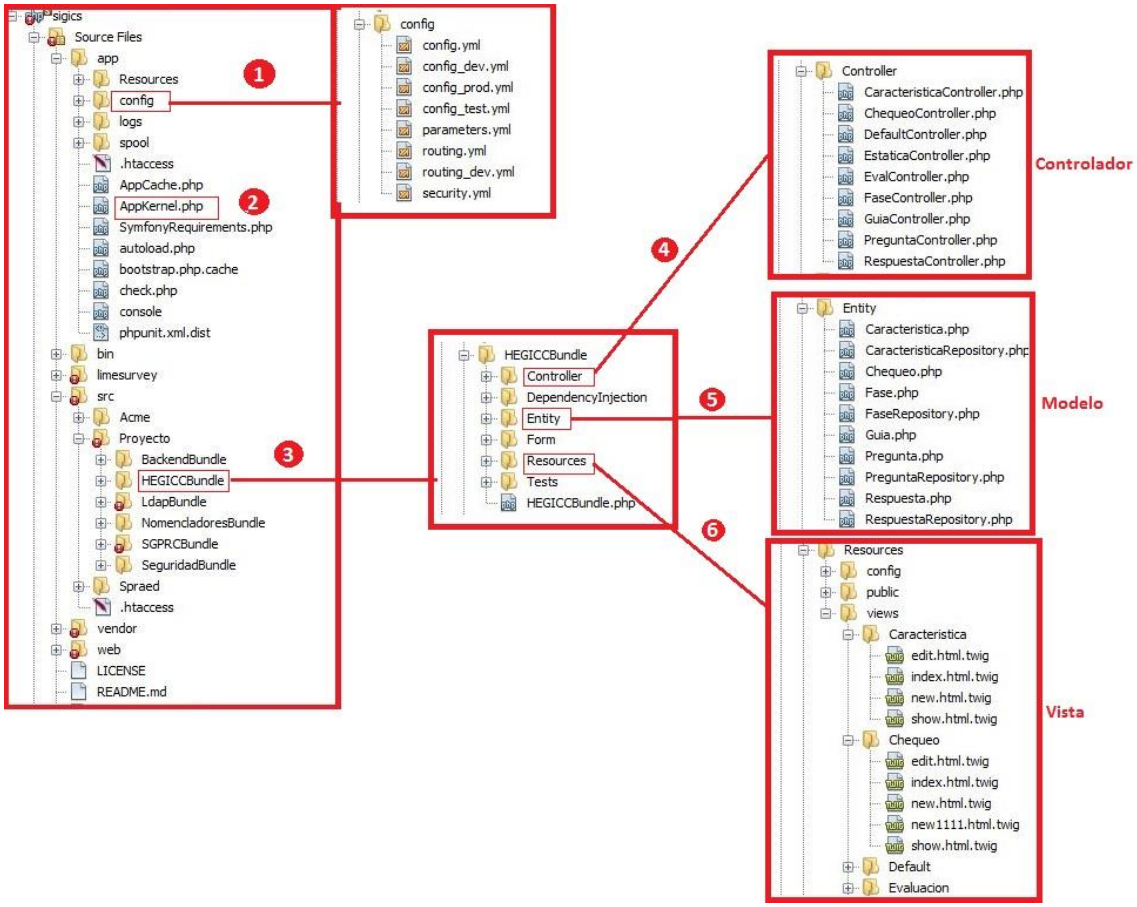

Fig. 3 Estructura de Clases

Symfony 2 posee un controlador frontal el cual es el encargado de crear el kernel de la aplicación mediante una instancia de la clase **(2)** y es la responsable de toda la configuración. Ofrece las rutas de los módulos o bundles que el usuario necesita para satisfacer su necesidad y que se encuentran en la carpeta config **(1)**. Es el núcleo de Symfony 2 y por tanto uno de los componentes fundamentales para su correcto funcionamiento. Dentro de la estructura del proyecto se puede observar la carpeta limesurvey la cual contiene el sistema para evaluar la satisfacción del cliente a través de encuestas. Dentro de la carpeta src se encuentra la carpeta Proyecto la cual posee los bundles de la aplicación y la carpeta Spraed que permite generar documentos en formato pdf. El sistema está organizado en cinco bundles o módulos fundamentales: SGPRC encargado del módulo revisiones del sistema, HEGICC encargado del módulo características de calidad, Seguridad posee el módulo de administración, Ldap para la autenticación de usuarios de la universidad y Nomencladores que posee todos los

nomencladores del sistema; estos se pueden observar dentro de la estructura de clases mostrada en la Fig 3. Los módulos poseen una organización común, la cual se muestra en **(3)** dentro de la misma se localizan las carpetas con los componentes específicos de la arquitectura:

Controller **(4)**: posee los ficheros con los códigos de las clases Controladoras.

Entity **(5)**: en esta carpeta se encuentran las Entidades del Modelo.

Resources **(6)**: dentro de esta se encuentran las carpetas con los CSS, los JavaScript y las Twig que conforman la Vista.

2.5.2 Tarjetas Clases Responsabilidad Colaborador (CRC)

Las tarjetas CRC fueron la base para la obtención del modelo entidad relación. Cada tarjeta se convirtió en objeto, sus responsabilidades en métodos públicos y sus colaboradores en llamadas a otras clases. Las tarjetas CRC constituyen una primera aproximación a los objetos que luego se van a utilizar en el desarrollo del sistema. En XP el proceso de diseño es iterativo por lo que la creación de las tarjetas no es en un mismo tiempo, se crean según las iteraciones y se le añaden responsabilidades y colaboradores según se haga necesario (Casas, y otros, 2009).

Estas tarjetas se dividen en tres secciones que contienen la información del nombre de la clase, sus responsabilidades y sus colaboradores.

Clase: nombre de la clase con que se está modelando.

Responsabilidades: las responsabilidades de una clase son las acciones que conocen y realizan, sus atributos y métodos.

Colaboradores: los colaboradores de una clase son las demás clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades.

A continuación se muestran las tablas 1, 2 y 3 ejemplos de tarjetas CRC confeccionadas durante la fase de diseño.

Módulo	
Responsabilidades • Crear Módulo • Editar Módulo • Mostrar Módulo • Eliminar Módulo • Listar Módulo	**Colaboradores** • Proyecto

Tabla 8 Tarjeta CRC Módulo

Pregunta	
Responsabilidades • Crear Pregunta • Editar Pregunta • Mostrar Pregunta • Eliminar Pregunta	**Colaboradores** • Característica • Fase

Tabla 9 Tarjeta CRC Pregunta

Plan de Revisiones	
Responsabilidades • Crear Plan de Revisiones • Editar Plan de Revisiones • Mostrar Plan de Revisiones • Eliminar Plan de Revisiones • Listar Plan de Revisiones	**Colaboradores** • Proyecto • Módulo • Disciplina

Tabla 10 Tarjeta CRC Plan de revisiones

2.5.3 Modelo de datos

Un modelo de datos es una serie de conceptos que puede utilizarse para describir los datos de acuerdo con reglas y convenios predefinidos y luego ser manipularlos (Jiménez, 2011). Se utiliza para describir la estructura lógica y física de la información persistente gestionada por el sistema, así como la correlación entre las clases de diseño y las estructuras de datos persistentes. En otras palabras, permite describir las estructuras de datos de la base de datos, su tipo, descripción y la forma en que se relacionan. A continuación se presenta una parte del modelo de datos del sistema por ser un modelo bastante amplio. En el anexo 2 se puede visualizar el modelo de datos.

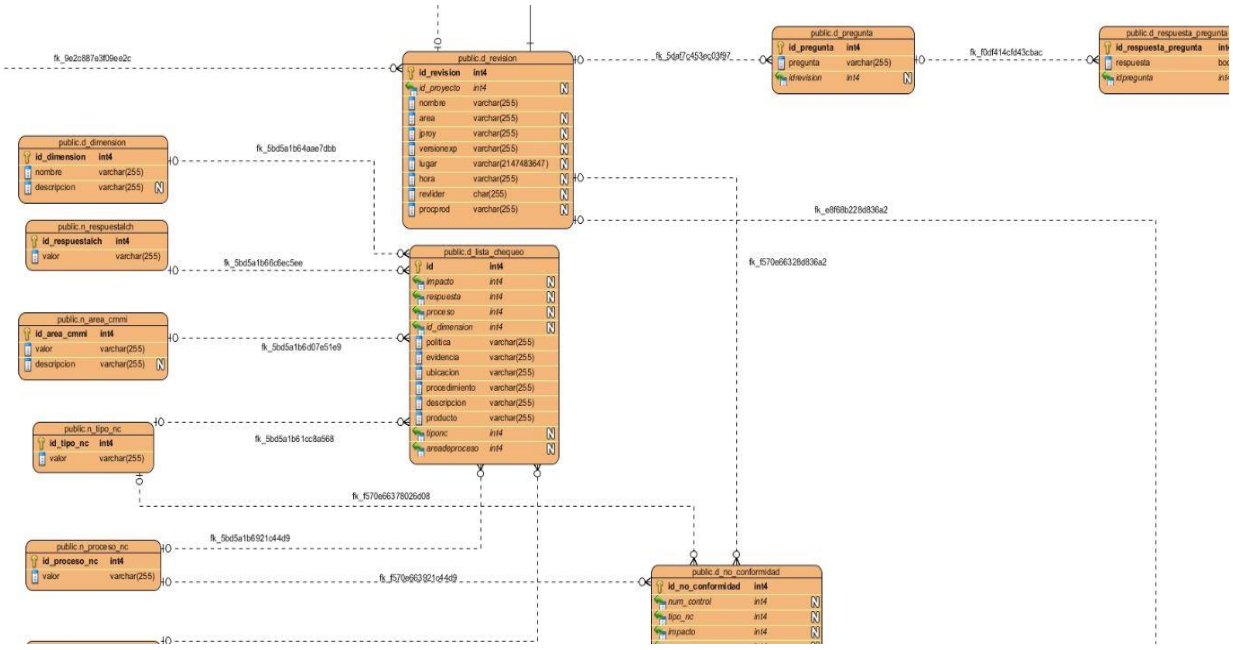

Fig. 4 Modelo de datos

2.5.4 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y proveen facilidades para crear un software reutilizable de buena calidad. Cada patrón describe un problema que ocurre repetidamente en el entorno, y describe el núcleo de la solución a ese problema, de tal forma que esta pueda ser usada un millón de veces, sin hacer el mismo trabajo dos veces (Asenjo González, y otros, 2003). A continuación se muestran los patrones utilizados en la solución.

2.5.4.1 Patrón de Base de datos

Los patrones de diseño de bases de datos son plantillas que ya han sido evaluadas como las responsables de resolver un problema, son la guía para apoyarse en la realización del trabajo. En esencia, los patrones de diseño de bases de datos constituyen la base para la búsqueda de soluciones a problemas comunes en el proceso de diseño de las mismas. (Blaha, 2010)

- ✓ **Patrón de llave subrogada:** este patrón consiste en asignar una llave o identificador único para cada entidad cuyo único requisito es almacenar un valor numérico único para cada fila de la tabla, actuando como una clave sustituta, de forma totalmente independiente a los datos de negocio (GuilleSQL, 2008). En la siguiente figura se puede visualizar un ejemplo del mismo.

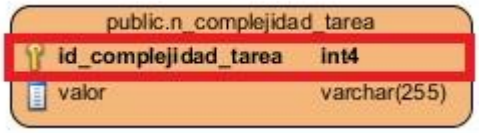

Fig. 5 Patrón de llave subrogada

2.5.4.2 Patrones de diseño GRASP

La asignación de responsabilidades es la habilidad más importante en el análisis y diseño orientado a objetos. Respetar los principios fundamentales es uno de los factores críticos para obtener diseños reutilizables, mantenibles y extendibles.

Experto: asigna la responsabilidad de la creación de un objeto o la implementación de un método a la clase que conoce toda la información necesaria para crearlo. Así se obtiene un diseño con mayor cohesión (Viscoti, y otros, 2004). Se evidencia en las clases modelos, ejemplo: la clase Usuario, la cual contiene toda la información referente a la manipulación de los datos de los usuarios.

Creador: el patrón creador permite identificar quién debe ser el responsable de la instanciación de nuevos objetos o clases (Viscoti, y otros, 2004). Da soporte al bajo acoplamiento. En Symfony2 en la clase Controller se definen y ejecutan un conjunto de acciones en las que se crean los objetos de las clases que representan las entidades, evidenciando de este modo que la clase Controller es creador de dichas entidades.

Bajo acoplamiento: consiste en tener las clases lo menos relacionadas posible, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en el resto de las clases (Viscoti, y otros, 2004). Esta característica permite potenciar la reutilización y disminuye la dependencia entre las clases. En el sistema la clase *PlanRevisionesController* al acceder a las entidades a través del método *getRepository*, asegura que exista un grado moderado de acoplamiento entre las clases pues no existen dependencias directas entre ellas.

Alta cohesión: se encuentra evidenciado en la implementación de las clases que forman parte de la capa del Modelo, las cuales están formadas por diferentes funcionalidades que se encuentran estrechamente relacionadas (Informáticas, 2014). Ejemplo: la entidad *NoConformidad* contiene varias funcionalidades estrechamente relacionadas con los datos que maneja.

Controlador: sirve como intermediario entre las capas presentación y acceso a datos. Ellas son las encargadas de ejecutar las funcionalidades para dar respuesta a la petición

del cliente. Es un objeto no destinado al usuario que se encarga de manejar un evento del sistema (Viscoti, y otros, 2004). En Symfony2 todas las peticiones web son manejadas por el controlador frontal, que es el único punto de entrada de toda la aplicación en un entorno determinado, el cual se encuentra en la carpeta web de cada proyecto de Symfony2. También este patrón se evidencia en la solución a través de las clases controladoras que se encuentran en la carpeta Controller perteneciente a cada bundle de la solución.

2.5.4.3 Patrones de diseño GOF

Patrones Estructurales: los patrones estructurales se ocupan, de cómo las clases y objetos se combinan para formar grandes estructuras y proporcionar nuevas funcionalidades.

- ✓ Fachada: provee de una única interfaz para acceder a un sistema completo, que actúa como único punto de acceso al mismo, y hace que este sea más fácil de utilizar. Ejemplo de esto se evidencia en la clase *PlanRevisionesController* que es la encargada del acceso a los datos de la entidad plan_revisiones en la BD.
- ✓ Decorador: es aplicado a la generación de vistas**,** la solución que ofrece dicho patrón es la de adicionar funcionalidad adicional a las plantillas. Por ejemplo añadir el menú y pie de página a las plantillas que lo requieran, se trata de decorar las plantillas con elementos adicionales reutilizables.

2.6 Codificación de la solución

En esta fase se genera todo el código fuente necesario para satisfacer las HU definidas para la solución. Al inicio de cada iteración, se lleva a cabo una revisión del plan de iteraciones y se modifica de ser necesario. Todas las HU son traducidas en tareas de programación.

Para llevar a cabo la correcta implementación de las HU se deben definir por parte del equipo de desarrollo las Tareas de Ingeniería (TI) que se realizan en cada una de las iteraciones. Las TI también conocidas como tareas de implementación permiten a los desarrolladores obtener un nivel de detalle más avanzado que el que propician las HU. A continuación se describe una de las tareas de ingeniería perteneciente a la primera iteración.

Tarea de Ingeniería	
Número Tarea:1	**Historia de Usuario:** 1, Autenticar Usuario
Nombre Tarea: Crear la interfaz para la autenticación.	
Tipo de Tarea: Desarrollo. **(Desarrollo/Corrección/Mejora)**	**Puntos Estimados(días):** 1
Fecha inicio:2/03/2015	**Fecha Fin:** 2/03/2015
Programador Responsable: Ebrik René López Ramírez, Anabel Fé León Mendoza	
Descripción: En esta tarea se crea la interfaz que permitirá autenticarse en el sistema.	

Tabla 11 Tarea de Ingeniería

Para una correcta comprensión y ejecución de la codificación resulta imprescindible el uso de estándares de codificación.

2.6.1 Estándares de codificación

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

- ✓ Todas las nomenclaturas a utilizar se definen en idioma español.
- ✓ Los identificadores para las variables y los parámetros se establecen con letra minúscula y en caso de ser un nombre compuesto se escriben juntos y de la segunda palabra en adelante se escriben con letra inicial mayúscula.

```
private $estadoEvalProy;
private $totalNc;
private $observaciones;
private $idRevision;
```

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

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

```
class CitaReunionController extends Controller {
```

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

```
class MinutaType extends AbstractType
```

- ✓ Las funciones deben ser llamadas sin espacios entre el nombre de la función, el signo de paréntesis y el primer parámetro, espacios entre cada coma por parámetro y sin espacios entre el último paréntesis, el signo de paréntesis cerrado y el signo de punto y coma.

```
$proyectos = $em->getRepository('SGPRCBundle:Proyecto')->findAll();
$entities = $em->getRepository('SGPRCBundle:CitaReunion')->findAll();
```

- ✓ Hacer uso de llaves para ganar en claridad del código.

```
public function indexAction() {
    $em = $this->getDoctrine()->getManager();
    $proyectos = $em->getRepository('SGPRCBundle:Proyecto')->findAll();
    $entities = $em->getRepository('SGPRCBundle:CitaReunion')->findAll();

    return $this->render('SGPRCBundle:CitaReunion:index.html.twig', array(
                'entities' => $entities,
                'proyectos' => $proyectos,
    ));
}
```

- ✓ Cadenas de texto entre comillas: PHP tiene dos formas de poner strings o cadenas de texto, con comillas simples y comillas dobles. La diferencia es que si se usa comillas dobles y se coloca dentro del texto un nombre de variable, el compilador lo interpretará y reemplazará por su valor. Por esta razón siempre se va a usar comillas simples a menos que se necesite hacer la interpolación de variables que permiten las dobles. Hay casos especiales donde es mejor usar dobles comillas (como cuando se usan caracteres de escape \ intensivamente).

```
$entity = $em->getRepository('SGPRCBundle:CitaReunion')->find($id);
```

```
<input type="hidden" name="_method" value="PUT" />
```

Los estándares de codificación permiten establecer un estilo de programación homogéneo permitiendo que cualquier persona que consulte el código lo pueda entender en menos tiempo.

2.7 Conclusiones parciales

Al finalizar el presente capítulo se arriba a las siguientes conclusiones:

- ✓ Después de realizar el análisis del sistema en términos de solución, quedaron definidas las Historias de Usuarios proporcionando una comprensión detallada de las funcionalidades de la aplicación.
- ✓ La propuesta de arquitectura del sistema se sustenta en un conjunto de componentes reutilizables que tienen como base el patrón arquitectónico MVC, lo que conforma un sistema flexible a cambios.
- ✓ La obtención de los requisitos funcionales y no funcionales, permitió definir el comportamiento y restricciones del sistema para su implementación.
- ✓ El empleo de patrones de diseño garantizó una solución que tiene como premisa la reutilización de código durante la fase de implementación del software.

CAPÍTULO 3: VALIDACIÓN DE LA INVESTIGACIÓN

3.1 Introducción

En este último capítulo se pretende verificar y validar el correcto funcionamiento del sistema, la verificación se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica y en la validación se ejecutan un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente. Para la comprobación final del sistema se aplican técnicas de validación de requisitos y métricas de software, además se realizan pruebas funcionales, de integración y pruebas de aceptación.

3.2 Validación de los requisitos

La validación de los requisitos tiene como objetivo comprobar que estos son correctos. Esta fase debe realizarse o de lo contrario se corre el riesgo de implementar una mala especificación, con el costo que eso conlleva. Es muy importante asegurar la validez de los requisitos antes de comenzar el desarrollo del software. Para ello debe hacerse una comprobación de la correspondencia entre las descripciones iniciales y la definición de los requisitos realizada, para verificar que responden a lo que desea el usuario final. Para llevar a cabo este proceso, se aplicaron las siguientes técnicas de validación de requisitos:

- ✓ Revisión de requisitos: se realizaron reuniones para detectar errores en el documento, donde se agregaron requisitos y se modificaron otros.
- ✓ Generación de casos de prueba de aceptación: para validar los requisitos funcionales de la solución, se diseñaron casos de pruebas de aceptación para cada una de las Historias de Usuario.

3.2.1 Métricas para requisitos

Los requisitos del software son la base de las medidas de la calidad. En la disciplina Requisitos se tuvo en cuenta la métrica para medir su estabilidad, especificidad y grado de validación.

Aplicación de la métrica Estabilidad de requisitos:

Teniendo en cuenta que se identificaron un total de 134 requisitos funcionales, de los cuales 10 resultaron modificados (3 por inserción, 2 por actualización y 5 por eliminación), se calcula:

ETR = [(134– 10) / 134] * 100= 92,53

Como resultado se obtuvo un valor de 92.53. Dicha cifra demuestra que no se han realizado cambios significativos sobre los requisitos, son estables y, por tanto, es confiable el análisis y diseño sobre ellos.

Aplicación de la métrica Especificidad de los requisitos:

La especificidad de los requisitos se calcula como:

Q1 = nui /nr = 134 /134= 1

Como resultado se obtuvo que la especificación de los requisitos no presenta ambigüedad, asegurando un alto nivel de calidad en el proceso de especificación.

Aplicación de la métrica Grado de validación:

El grado de validación de los requisitos se calcula:

Q3 = nc / (nc + nnv) = 134 /(134+ 0) = 1

La aplicación de esta métrica dio como resultado 1, por lo tanto se concluye que la definición de los requisitos es correcta.

3.3 Validación del diseño

Para comprobar la calidad del diseño del sistema se emplearon las métricas Relaciones entre Clases (RC) y Tamaño Operacional de Clase (TOC).

Relaciones entre clases (RC)

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

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

Para determinar el grado de afectación de los atributos de calidad que mide la métrica RC es necesario determinar la cantidad de relaciones de uso (CRU) que posee cada una de las clases a medir. Una vez determinada la CRU, se procede a calcular el promedio de las mismas y teniendo ambos valores según los criterios expuestos en el capítulo 1 se determina la incidencia de los atributos de calidad en cada una de las clases.

La aplicación del instrumento de evaluación de la métrica RC para el número total de relaciones arrojó los resultados que se plasman en el gráfico de la figura.

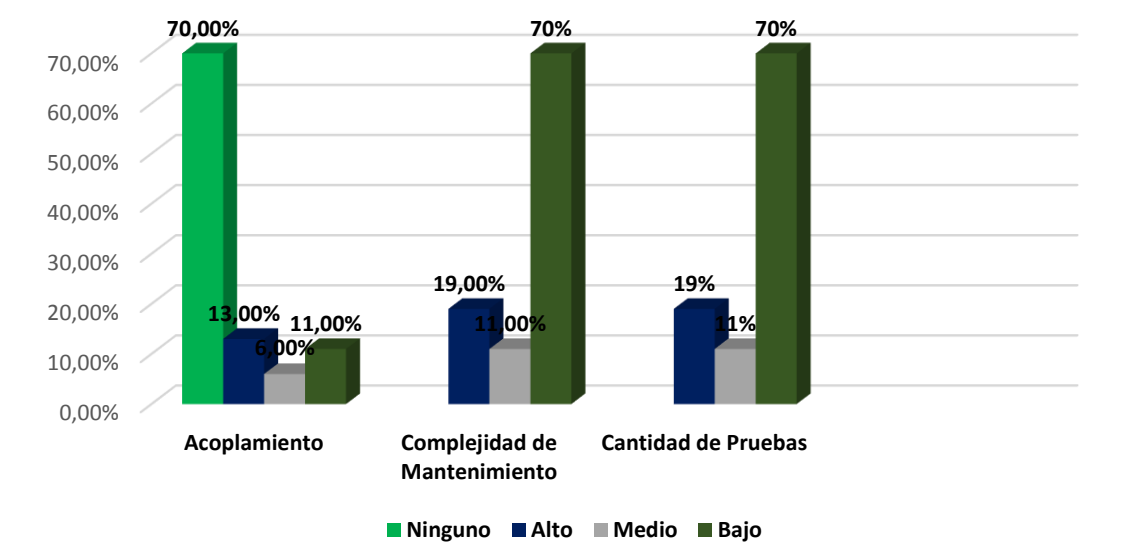

Fig. 6 Resultado de la métrica RC

Acoplamiento: según los resultados que se muestran, el (70%) de las clases no posee relaciones de uso por lo que no tienen valores de acoplamiento, validando una realización correcta del diseño.

Complejidad de mantenimiento: según los resultados que se muestran en la figura, el 70% de las clases se comportan de forma satisfactoria pues son de fácil soporte.

Cantidad de pruebas: luego de aplicar la métrica se obtuvo que el (70%) de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones y pruebas de software.

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

Tamaño operacional de la clase (TOC)

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

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

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

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

Para determinar el valor de los atributos de calidad, se debe determinar la cantidad de procedimientos (CP) que posee cada una de las clases a medir. Una vez determinado el CP se procede a calcular el promedio del mismo y según los criterios expuestos en el capítulo 1 se determina la incidencia de los atributos de calidad en cada una de las operaciones de las clases.

La aplicación del instrumento de evaluación de la métrica TOC para el número total de operaciones arrojó los resultados que se plasman en el gráfico de la figura.

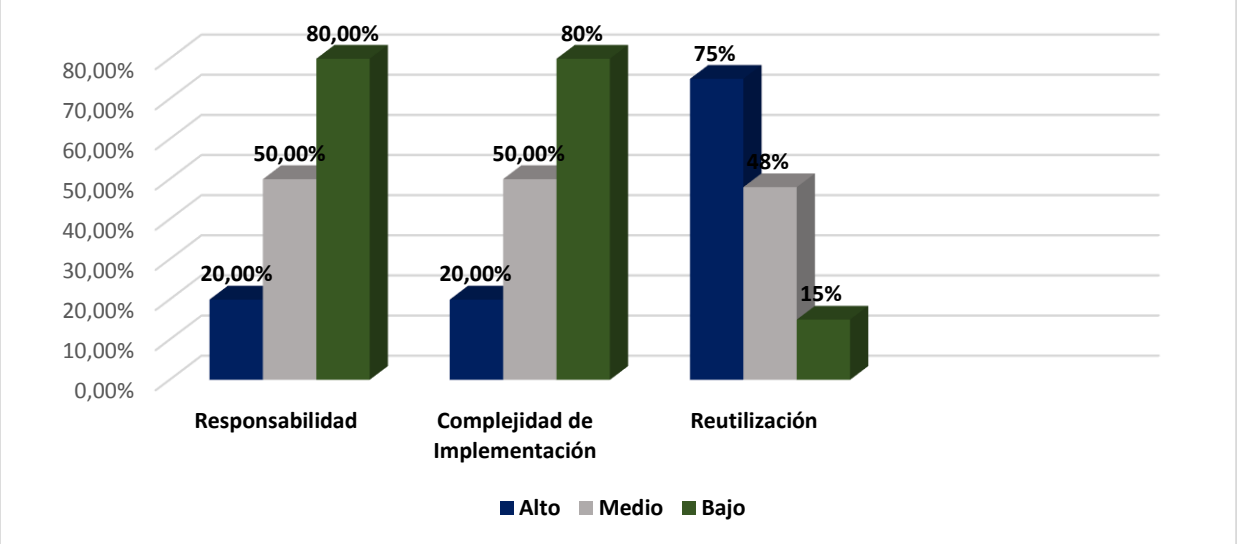

Fig. 7 Resultado de la métrica TOC

Responsabilidad: luego de aplicar la métrica se obtuvieron resultados satisfactorios que reflejan una responsabilidad baja con valor del 80%.

Complejidad de Implementación: después de haberse realizado la medición de la métrica, arrojó también resultados positivos ya que la complejidad de las clases es baja en un 80%.

Reutilización: se obtuvieron valores que según muestra la gráfica de la figura anterior se comporta en un nivel alto con un 75%.

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

3.4 Verificación del sistema

Las pruebas tienen como objetivo valorar y mejorar la calidad de los productos del trabajo generado durante el desarrollo y modificación del software. Según (Pressman, 2000) verificación *es el conjunto de actividades que aseguran que el software implemente correctamente una función específica, y la* validación *es un conjunto diferente de actividades que aseguran que el software construido corresponde y satisface los requisitos del cliente.*

3.4.1 Nivel de Unidad

Las pruebas a este nivel son aplicadas para verificar que el software cumple los requisitos funcionales y también son empleadas para asegurar la calidad del código entregado. Además, son la mejor forma de detectar fallas tempranamente en el desarrollo y está demostrado que mientras más pronto se encuentren los errores, menos costará corregirlos. Se realizan pruebas de funcionalidad, utilizando los métodos de prueba de caja blanca y caja negra para comprobar que tanto el código como la interfaz no contengan errores y se ejecutan adecuadamente.

3.4.1.2 Método de pruebas de caja negra

Para la realización de las pruebas de caja negra se empleó la técnica partición de equivalencia que garantizó efectividad al examinar los valores válidos e inválidos de las entradas existentes en el software. A continuación se brinda un ejemplo de un caso de prueba de uno de los requisitos que componen la funcionalidad Adicionar Proyecto, el resto de estos se incluyen en el documento 0122_Diseño de Casos de Pruebas basado en requisitos.

Escenario	Descripción	Nombre	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Agregar proyecto correctamente.	Permite agregar el proyecto con sus datos correspondientes.	V Fiscalía	V Realizar una Jurisdicción Voluntaria y un proceso Ordinario.	El proyecto ha sido adicionado correctamente.	Seleccionar el módulo Administración en el menú superior . Seleccionar de la lista desplegable Gestionar Proyecto y en la parte lateral izquierda seleccionar la opción Agregar Proyecto.
EC 1.2 Agregar proyecto con campos incompletos.	El sistema no permite agregar el proyecto porque existen campos obligatorios vacíos.	I V Fiscalía	V Realizar una Jurisdicción Voluntaria y un proceso Ordinario. N/A	El sistema muestra un mensaje de error informando que existen campos que son obligatorios vacíos.	
EC 1.3 Agregar proyecto con campos incorrectos.	El sistema no permite agregar el proyecto porque existen campos incorrectos.	I fiscalía V Fiscalía	V Realizar una Jurisdicción Voluntaria y un proceso Ordinario. I Realiz$r una Jurisdic*ión Voluntaria y un proceso Ordinari@.	El sistema muestra un mensaje de error informando que existen campos que son incorrectos.	

Descripción de las variables

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de texto	No	El campo es obligatorio y está representado con * en el sistema, admite números y letras.
2	Descripción	Campo de texto	Sí	Campo no obligatorio, admite números y letras.

Fig. 8 Caso de Prueba Adicionar Proyecto

Luego de aplicar las pruebas de caja negra por el Grupo de Calidad del centro CEGEL se obtuvieron los siguientes resultados:

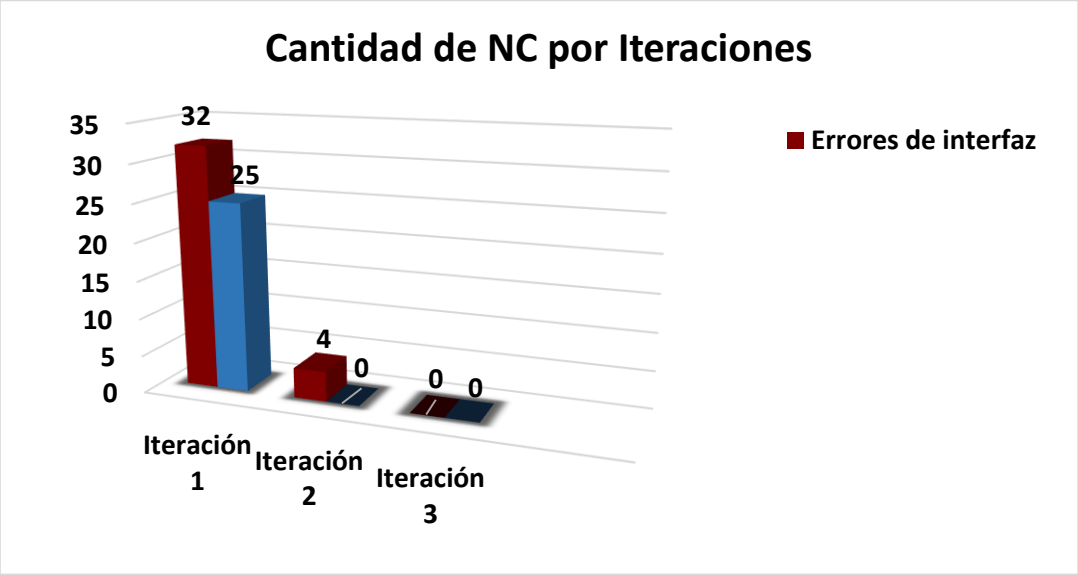

Fig. 9 Errores detectados en las pruebas funcionales

Las no conformidades encontradas en la primera y segunda iteración estaban relacionadas con la validación de los datos que se introducen en el sistema y errores en las interfaces, pudiendo comprobar en la tercera iteración realizada que estos fueron corregidos y que la aplicación desarrollada funciona correctamente.

Después de concluidas las pruebas se liberó la aplicación SIGICS (Sistema de Gestión Integral de la Calidad de Software) entregándose al equipo de desarrollo el Acta de Liberación Interna de Productos de Software en la que consta que la aplicación está apta para ser utilizada Ver Anexo 3.

3.4.1.3 Método de pruebas de caja Blanca

Para aplicar las pruebas de caja blanca se empleó la técnica del camino básico. Esta permitió obtener una medida de la complejidad lógica para el diseño de los casos de pruebas y usar dicha medida como guía para la definición de un conjunto básico de caminos de ejecución. Se tomó como ejemplo el método *updateAction ()*, perteneciente a la clase *CaracteristicaController*. Para ello se procede a enumerar las sentencias del código y a partir del mismo se construye el grafo de flujo asociado.

```
    public function updateAction(Request $request, $id) {
1       $em = $this->getDoctrine()->getManager();
        $proyectos = $em->getRepository('SGPRCBundle:Proyecto')->findAll();
        $entity = $em->getRepository('HEGICCBundle:Caracteristica')->find($id);

2       if (!$entity) {
3           throw $this->createNotFoundException('Unable to find Caracteristica entity.');
        }

4       $deleteForm = $this->createDeleteForm($id);
        $editForm = $this->createForm(new CaracteristicaType(), $entity, array(
            'action' => $this->generateUrl('caracteristica_update', array('id' => $entity->getId())),
            'method' => 'PUT',
        ));
        $editForm->handleRequest($request);

        $existe = $em->getRepository("HEGICCBundle:Caracteristica")->findOneByOrden($entity->getOrden());

5       if (!$existe || $existe == $entity) {
7           if ($editForm->isValid()) {
                $em->flush();
8               $this->get('session')->getFlashBag()->add('info', "La entidad ha sido modificada correctamente.");
                return $this->redirect($this->generateUrl('caracteristica', array('id' => $id)));
            }
        } else {
6           $this->get('session')->getFlashBag()->add('error', "El orden especificado ya ha sido asignado.");
        }
```

```
9       return array(
            'entity' => $entity,
            'proyectos' => $proyectos,
            'edit_form' => $editForm->createView(),
            'delete_form' => $deleteForm->createView(),
        );
    }
```

Fig. 10 Código fuente del método updateAction

Partiendo del método antes mencionado se obtuvo el siguiente grafo de flujo.

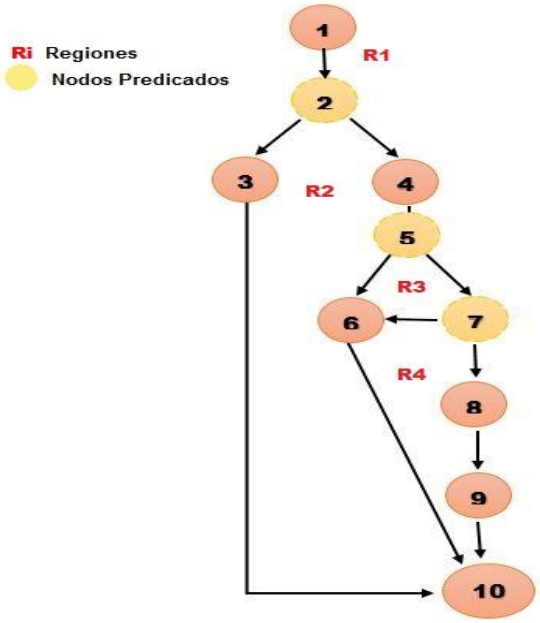

Fig. 11 Grafo de flujo del método updateAction

Luego de haber realizado la construcción del grafo de flujo se procede a calcular la complejidad ciclomática mediante tres fórmulas que se describen a continuación, las cuales deben exponer el mismo resultado para asegurar que el cálculo de la complejidad es correcto. Cada fórmula V (G) representa el valor del cálculo.

V (G) = (A - N) + 2

Donde A es el número de aristas y N es el número de nodos contenidos en el grafo.

V (G) = (12 - 10) + 2

V (G) = 4

V (G) = P + 1

Donde P es el número de nodos predicados contenidos en el grafo. Los nodos predicados son aquellos que parten de dos o más aristas.

V (G) = 3 + 1

V (G) = 4

V (G) = R

Donde R es la cantidad total de regiones.

V (G) = 4

El número de regiones del grafo es igual a la complejidad ciclomática.

El cálculo efectuado anteriormente dio como resultado el mismo valor en todos los casos, la complejidad ciclomática es de 4, este valor indica que existen 4 posibles caminos por donde el flujo puede circular y determina el número de casos de prueba que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. A continuación se representan los caminos básicos por los que puede transitar el flujo:

- Camino 1**:** 1 - 2 - 3 - 10
- Camino 2: 1 - 2 - 4 - 5 - 6 - 10
- Camino 3: 1 - 2 - 4 - 5 - 7 - 6 - 10
- Camino 4: 1 - 2 - 4 - 5 - 7 - 8 - 9 - 10

Luego de establecidos los caminos básicos se procede a realizar los casos de prueba para cada uno de ellos, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino. A continuación se presenta un ejemplo del caso de prueba para el camino 4 realizado a esta funcionalidad.

Entrada	Que no exista la característica que se va a editar, que el orden especificado no exista y que el formulario sea válido.
Resultados esperados	Redirecciona a la página edit.html.twig.
Condiciones	$entity==false, $existe==false, $existe == $entity, $editForm->isValid()==true

Tabla 12 Caso de prueba para el camino 4

Una vez ejecutados todos los casos de pruebas obtenidos a través de la aplicación de la técnica camino básico se concluye que los mismos fueron probados satisfactoriamente demostrando que el código generado no presenta ciclos infinitos y no existe código innecesario en el sistema desarrollado.

3.4.2 Nivel de Integración

Las pruebas de integración prueban la interacción entre dos o más elementos, que puede ser clases, módulos, paquetes, subsistemas, etc, incluso la interacción del sistema con el entorno de producción. El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes (Pruebas de integración, 2012).

La integración puede ser descendente si se integran los módulos desde el programa principal, o bien, ascendente, si la verificación del diseño empieza desde los módulos más bajos y de allí al principal.

Para llevar a cabo las pruebas de integración se hizo uso de la integración descendente ya que se integran los módulos, comenzando por el módulo principal en este caso el módulo Administración, para luego ir incorporando los módulos subordinados (Revisiones, Características de Calidad y Satisfacción del Cliente). Para ello el equipo de desarrollo realizó varias pruebas cada vez que se integraba un nuevo módulo. Obteniendo un total de 23 no conformidades al producto realizado por parte del equipo de trabajo, descomponiéndose en 10 no conformidades en el módulo de Revisiones, 5 en el módulo Características de Calidad, 5 en Satisfacción del Cliente y 3 en el módulo Administración, erradicando estas no conformidades y realizar una correcta integración.

3.5 Validación del sistema

Con la validación del sistema se pretende comprobar que el software cumple las expectativas que el cliente espera. Para llevar a cabo la validación de la aplicación se aplicaron las pruebas de aceptación por cada Historia de Usuario definida en la etapa de planificación.

3.5.1 Nivel de aceptación

Las pruebas de aceptación son destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente final. Estas pruebas aseguran el comportamiento del sistema y especifican los aspectos a probar cuando una Historia de Usuario ha sido correctamente implementada (Riva, 2014).

A cada una de las HU se le realizaron pruebas de aceptación. Estas pruebas son reflejadas mediante casos de pruebas de aceptación, los cuales están conformados por

7 parámetros, que dan información acerca de la prueba realizada. Los parámetros a medir son:

- ✓ Código: muestra un identificador para cada prueba realizada (normalmente se pone el nombre del componente seguido de las letras PA (pruebas de aceptación).
- ✓ Nombre de la Historia de Usuario: indica el nombre de la prueba.
- ✓ Descripción: se describe cuál es la funcionalidad que se va a medir del componente al que se le esté realizando la prueba.
- ✓ Condiciones de ejecución: indica cuáles son las condiciones que se tienen que cumplir para que el componente realice correctamente la funcionalidad que se va a medir.
- ✓ Entrada / Pasos de ejecución: indica los pasos a seguir para realizar la prueba.
- ✓ Resultados esperados: muestra cuál es el resultado que se obtendría de un correcto funcionamiento de la prueba.
- ✓ Evaluación de la prueba: indica el estado de la prueba si es satisfactoria o insatisfactoria.

A continuación se muestra un ejemplo de caso de prueba de aceptación:

Caso de prueba de aceptación	
Código:PA3-HU03	**Historia de Usuario:** Autenticar usuario
Nombre: Autenticar usuario.	
Descripción: para acceder al sistema el usuario debe de autenticarse previamente.	
Condición de ejecución: el usuario debe formar parte de los usuarios del dominio uci.	
Entrada/Pasos de ejecución: ✓ El usuario introduce su usuario. ✓ El usuario introduce su contraseña. ✓ El usuario presiona el botón Entrar.	
Resultado esperado: en caso que se introduzca algún dato incorrecto se mostrará el siguiente mensaje: Se ha producido un error, verifique que lo datos introducidos son correctos. Si todo está correcto se accede a la aplicación sin problemas.	
Evaluación de la prueba: satisfactoria.	

Tabla 13 PA Autenticar usuario

3.6 Validación de las variables de la investigación

Con el desarrollo del sistema, se contribuye a la mejora del proceso de gestión de la calidad para el Centro de Gobierno Electrónico, facilitando la ejecución de las actividades del mismo y disminuyendo el tiempo de recolección y análisis de los datos del área, esto se evidencia con los siguientes elementos:

Antes	Después
Los usuarios accedían a cada uno de los sistemas por separado, autenticándose más de una vez, ya que intervenían en la ejecución de los distintos procesos que se llevaban a cabo en el Grupo de Calidad.	Actualmente los usuarios acceden a un único sistema, permitiéndoles realizar los diferentes procesos que se llevan a cabo en el Grupo de Calidad de forma completa.
Cada uno de los sistemas funcionaban de forma aislada y algunas funcionalidades eran similares, retrasando las actividades a desarrollar por el Grupo de Calidad. Además de utilizar bases de datos diferentes en la que existe información repetida.	El sistema unifica anteriores funcionalidades similares en un módulo de Administración, disminuyendo el tiempo de ejecución de los diferentes procesos. Además utiliza una única base de datos, eliminando la duplicidad de información que generaban estos procesos por separados.

Tabla 14 Validación de las variables

Por los elementos antes expuestos se demuestra el cumplimiento del problema de investigación planteado, comprobando con el desarrollo del sistema realizado, que se contribuye a la disminución del tiempo de recolección y análisis de los datos del área.

3.7 Conclusiones parciales

Al concluir el presente capítulo se evidencia que:

- ✓ Se aplicaron técnicas de validación de requisitos permitiendo asegurar la validez de los mismos en el proceso de desarrollo de software.
- ✓ La aplicación de las métricas para requisitos arrojó como resultado que los mismos son estables y por tanto, es confiable el diseño efectuado sobre ellos.
- ✓ La aplicación de métricas de diseño demostró que las clases del diseño poseen bajo acoplamiento, que existe además una baja responsabilidad y complejidad de implementación y una alta reutilización en el diseño propuesto.
- ✓ Se aplicaron pruebas de funcionalidad utilizando los métodos de prueba de caja blanca y de caja negra lo que permitió verificar que el software cumple los requisitos funcionales.
- ✓ Se validaron las variables que forman parte del problema de la investigación, demostrando que el sistema desarrollado cumple con las necesidades del Grupo de Calidad del centro CEGEL.

CONCLUSIONES GENERALES

Con la culminación de la presente investigación se concluye que:

- ✓ El estudio de los referentes teóricos y de los sistemas de gestión de la calidad existentes demostró que las soluciones analizadas no se ajustan a las necesidades del Grupo de Calidad del centro CEGEL.
- ✓ Las tecnologías y herramientas seleccionadas, así como la utilización de XP como metodología de desarrollo, en correspondencia con los criterios de selección definidos, facilitaron la realización del Sistema de Gestión Integral de la Calidad de Software que dio solución a la problemática identificada.
- ✓ Con la utilización de patrones arquitectónicos y de diseño, se logró implementar un sistema de acuerdo a los estándares y modelos utilizados en el desarrollo de software que responden a las necesidades del cliente.
- ✓ Mediante el diseño y aplicación de los casos de prueba se logró valorar los resultados y verificar que las funcionalidades cumplieran con las restricciones definidas por el cliente.
- ✓ Como resultado general del trabajo realizado se le dio cumplimiento al objetivo general propuesto, implementándose el Sistema de Gestión Integral de la Calidad de Software para el centro CEGEL.

RECOMENDACIONES

Se recomienda incorporarle nuevos módulos al Sistema de Gestión Integral de la Calidad de Software (SIGICS) de acuerdo a nuevos procesos que se definan en el Grupo de Calidad.

Adaptar el sistema a los nuevos cambios que sufran los procesos definidos.

Bibliografía

Addison Wesley Helm, R., Johnson, R. and Vlissides, J. 1995. 1995.

Asenjo González, Diego Andrés and Ríos Peña, Alejandro. 2003. *Patrones de diseño.* Habana : s.n., 2003.

Beck, Kent. 1999. *Extreme Programming Explained. Pearson Education.* s.l. : Embrace Change sl, 1999.

Blaha, Michael,. 2010. *Patterns of Data Modeling.* 2010.

Calabria, Luis and Píriz, Pablo. 2003. *Metodología XP.* Universidad ORT Uruguay : s.n., 2003.

Callejas, Manuel Arias. 2009. Carmen.Estándares de codificación. [Online] Mayo 7, 2009. [Cited: Mayo 20, 2015.] http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf.

Casas, Sandra and Reinaga, Héctor. 2009. *Aspectos Tempranos: Un enfoque basado en Tarjetas CRC.* Argentina : s.n., 2009.

CSS3. 2015. Características de Css3. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.css3.com.

Doctrine. 2014. Doctrine. [Online] 2014. [Cited: Abril 14, 2015.] http://docs.doctrine-project.org/en/2.0.x/reference/introduction.html.

Eguiluz, Javier. 2010. Libros web. [Online] 2010. [Cited: Abril 16, 2015.] http://librosweb.es/libro/javascript/.

Federico Alonso Atehortúa Hurtado, Ramón Elias Bustamantes Vélez, Jorge Alberto Valencia de los Ríos. 2003. *Sistema de gestión integral. Una sola gestión,un solo equipo.* Madellín Colombia : Universidad de Antioquia, 2003. ISBN:978-958-714-158-0.

Felix, Sarisleidy y Angel. 2014. Tesis Sarisleidy y Angel Felix. [Online] 2014. [Cited: Febrero 9, 2015.] http://biblioteca.uci.cu/Catalogo/Descarga.

Gabriel Vargas, César y Biagioli, Germán. 2009. *Sistema para auditar el cumplimiento de CMMI-SW nivel 2.* 2009.

Gauchat, Juan Diego. 2012. *El gran libro de HTML5, CSS3 y JavaScript.* Barcelona : MARCOMBO, 2012.

Gespro, Xedro. 2003. Xedro gespro. *Suite de gestión de proyectos.* [Online] 2003. [Cited: Febrero 9, 2015.] http://gespro.cegel.prod.uci.cu/gespro/about.

GuilleSQL. 2008. GuilleSQL. [Online] Noviembre 18, 2008. [Cited: Marzo 24, 2015.] http://www.guillesql.es/Articulos/Claves_Subrogadas_Slowly_Changing_Dimension_SCD_Tipo_2.aspx.

HTML 5. Hipertextual. *Entendiendo HTML5: guía para principiantes.* [Online] [Cited: Abril 10, 2015.] http://hipertextual.com/archivo/2013/05/entendiendo-html5-guia-para-principiantes/.

Hurtado, Federico Alonso Atehortúa. 2009. *Sistema integrado de Gestión.* 2009.

IBM. 2015. Unified Modeling Language (UML. [Online] 2015. [Cited: Febrero 9, 2015.] http://www-01.ibm.com/software/rational/uml/.

IEEE. 2015. IEEE. [Online] 2015. [Cited: Abril 23, 2015.] http://www.computer.org/cms/Computer.org/Computer.org/s2esc/s2esc_pols/SP-06_Vocabulary_Objectives.htm.

Informáticas, Serie Científica de la Universidad de las Ciencias. 2014. Serie Científica de la Universidad de las Ciencias Informáticas. [Online] 2014. [Cited: Marzo 26, 2015.] publicaciones.uci.cu/index.php/SC/article/download/23/24.

ISO-9000. 2000. *Sistemas de gestión de la calidad en Fundamentos y vocabulario.* 2000.

—. 2000. *Sistemas de gestión de la calidad, en Fundamentos y vocabulario.* 2000.

Jiménez, Susana Alejandra López. 2011. ITSLR Instituto Tecnológico Superior de los Reyes. [Online] Agosto 22, 2011. [Cited: Mayo 7, 2015.] http://itslr-alelopj.weebly.com/uploads/9/3/6/4/936494/fbd-tutorial_u2.pdf.

Kmkey. 2015. Kmkey. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.kmkey.com/productos/software_gestion_calidad..

Labrada, Yaima. 2013. Tesis 32-Yaima Morales Labrada. [Online] 2013. [Cited: Febrero 9, 2015.] http://biblioteca.uci.cu/Catalogo/Descarga.

Lincke, R, J, Lundberg and W, Lowe. 2008. *Comparing software metrics tools. International Symposium on Software Testing and Analysis.* Berlin : s.n, 2008.

Lopez, C. 2000. Aseguramiento de las Características de Calidad. [Online] 2000. http://www.gestiopolis.com/canales/gerencial/articulos/27/ISO.htm.

Magdalena, Universidad de. 2010. SQA-Ingeniería de Software. [Online] 2010. [Cited: Abril 13, 2015.] https://sqaisoft.wordpress.com/sqa/aseguramiento-de-la-calidad-de-procesos-y-productos-en-cmmi-ppqa/.

Maza Oval, Doris and Arencibia Cruz, Humberto. 2013. *Diseño e implementación del procedimiento Ejecutivo Económico del Proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.* La Habana : s.n., 2013.

Métricas, Guía de. 2015. Propuesta de una guía de métricas. [Online] 2015. [Cited: Febrero 9, 2015.] http://vinculando.org/articulos/sociedad_america_latina/propuesta_guia_de_medidas_para_evaluacion_sistemas_informacion.html.

NetBeans. 2015. NetBeans IDE-Overview. [Online] 2015. [Cited: Febrero 9, 2015.] https://netbeans.org/features/index.html.

Orlando, Yarinet y. 2014. Defensa Yarinet y Orlando v1.4. [Online] 2014. [Cited: Febrero 9, 2015.] http://biblioteca.uci.cu/Catálogo/Descaga.

Pacheco, N. 2013 . Manual de Twig. [Online] 2013 . [Cited: Abril 16, 2015.] http://gitnacho.github.io/Twig/ .

Peña, Juan Manuel Fernández. 2010. Pruebas de software. [Online] Julio 2010. [Cited: Febrero 9, 2015.] www.uv.mx/personal/jfernandez/files/2010/07/Cap3-Caminos.pdf.

Pereiro, Jorge. 2001. *La satisfacción del cliente en ISO 9001.* 2001.

PHP, Manual. 2015. PHP: ¿Qué es PHP? - Manual. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.php.net/manual/es/intro-whatis.php.

PostgreSQL, Sobre. 2015. Sobre PostgreSQL. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.postgresql.org.es/sobre_postgresql.

Potencier, Fabien, Zaninotto, Francois. 2008. *Symfony, la guía definitiva.* 2008.

Pressman. 2000. *Ingeniería de Software. Un enfoque práctico. Pág. 384.* 2000.

—. 2000. *Software Engineering: A Practitioner´s Approach. Séptima edición.* New York, EEUU : McGraw-Hill, 2000. 870..

Pressman, Roger. 2010. *Ingeniería de Software un enfoque Práctico.* s.l. : ISBN: 978-0-07-337597-7, 2010.

—. 2000. *Ingeniería de software. Un enfoque práctico.* s.l. : McGraw-Hill, Nueva York, E.U.A. Capitulo 9, 2000.

—. 2002. *Ingeniería de software. Un enfoque práctico,5ta edición.* s.l. : McGraw-Hill, 2002. ISBN: 8448132149.

Pressman, Roger S. 2000. *Ingenieria de requisitos.Un enfoque practico ,6ta edición. Métricas del producto para el software.* 2000.

—. 2000. *Ingeniería de Software. Un enfoque práctico. 5ta edición.* 2000. 8448132149..

—. 2002. *Ingeniería del Software. Un enfoque práctico 6ta edición.* Nueva York : McGraw-Hill, 2002. 0072853182.

Programación, Lenguaje de. 2015. Lenguaje de Programación. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.programa.us/asesorias/educacion/musica/lenguaje_de_programacion.

Project, The Apache HTTP Server. 2015. Welcome! - The Apache HTTP Server Project. [Online] 2015. [Cited: Febrero 9, 2015.] http://httpd.apache.org.

Pruebas de integración. 2012. [Online] Octubre 28, 2012. [Cited: Mayo 26, 2015.] http://es.slideshare.net/pablis001/estrategias-de-aplicaciones-para-las-pruebas-de-integracin.

Rabí, Dayanis Elvia Alcantara and Luque, Eylín Hernández. 2010. Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas de Información Geográfica. [Online] Enero 4, 2010. [Cited: Febrero 9, 2015.] http://vinculando.org/articulos/sociedad_america_latina/propuesta_guia_de_medidas_para_evaluacion_sistemas_informacion.html.

—. 2010. Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas de Información Geográfica. [Online] Enero 4, 2010. [Cited: Febrero 9, 2015.] http://vinculando.org/articulos/sociedad_america_latina/propuesta_guia_de_medidas_para_evaluacion_sistemas_informacion.html.

Requerimientos, Metodología Gestión de. 2015. Metodología Gestión de Requerimientos. [Online] 2015. [Cited: Junio 10, 2015.] https://sites.google.com/site/metodologiareq/capitulo-ii/tecnicas-para-identificar-requisitos-funcionales-y-no-funcionales.

requisitos, Métricas para. 2015. Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas de Información Geográfica. [Online] 2015. [Cited: Febrero 9, 2015.] http://vinculando.org/?s=m%C3%A9tricas+para+requisitos.

Riva, José. 2014. Actas de los talleres de Ingeniería de Software y Bases de Datos. [Online] 2014. [Cited: Junio 2, 2015.] http://www.sistedes.es/TJISBD/Vol-3/No-4/PRIS09.pdf.

Sánchez Fornaris, Maite y Alcantara Rabí, Dayanis. 2009. *Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas.* 2009.

Servidores, Soporte a. 2015. Soporte a Servidores. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.ustamed.edu.co/sistemas/index.php/frentes/sistemas/soporte-a-servidores.

software, Instituto de Ingeniería de. 2010. *CMMI para desarrollo, Versión 1.3.* 2010. ECS-TR-2010-33.

Sommerville, Ian. 2005. *Ingeniería de software Séptima edición.* España(Madrid) : ISBN:84-7829-074-5, 2005.

Symfony. 2003. Web, Introducción a Symfony 2 | Maestros del WebMaestros del Web. [Online] 2003. [Cited: Febrero 9, 2015.] http://www.maestrosdelweb.com/editorial/curso-symfony2-introduccion-instalacion.

TIC, Las. 2015. AnetCom. Las TIC en la estrategia empresarial. [Online] Febrero 9, 2015. http://video.anetcom.es/editorial/Las_TIC_en la_estrategia_empresarial.pdf.

Torres, Fabio. 2012. Memorias dentro del desarrollo de software. [Online] Febrero 2012. [Cited: Marzo 23, 2015.] http://phigux.blogspot.com/2012/02/que-es-el-levantamiento-de.html.

Twig. 2012. TWIG, EL MOTOR DE PLANTILLAS PARA PHP. [Online] 2012. [Cited: Abril 10, 2015.] http://www.acens.com/wp-content/images/2014/06/twig-plantillas-wp-acens.pdf.

UML, Visual Paradigm for. 2015. Guión Visual Paradigm for UML. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/PracticaVP.pdf..

Velthuis, Mario G Piattini. 2003. *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión.* Madrid : RA-MA EDITORIAL, 2003.

Viscoti, Astudillo Marcello and Hernán. 2004. Fundamentos de Ingeniería de Software. [Online] Septiembre 9, 2004. [Cited: Marzo 24, 2015.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

Web, Desarrollo. 2015. Desarrollo Web. [Online] 2015. [Cited: Febrero 9, 2015.] http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html.

Zapata, Carlos Mario, Palacio, Carolina and Olaya, Natalí. 2007. Revista EIA UNC-ANALISTA: HACIA LA CAPTURA DE UN CORPUS DE REQUISITOS. [Online] Junio 2007. [Cited: Junio 10, 2015.] http://www.scielo.org.co/scielo.php?pid=S1794-12372007000100003&script=sci_arttext. ISSN 1794-1237.