

Universidad de las Ciencias Informáticas

Facultad 4



**TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE
INGENIERO EN CIENCIAS INFORMÁTICAS**

Título: Informatización del procedimiento de Revisiones Técnicas
Formales en el área Arquitectura del desarrollo de software para el centro
FORTES

Autores: Yairon Consuegra Cabrera
Gino Miguel Ricardo González

Tutores: Ing. Lisandra Peña Espinosa
Ing. Lizardo Ramírez Tabuada

Co-tutora: Ing. Roselia González Cardoso

La Habana, 2015

Declaración de autoría

Por este medio declaramos ser los autores del presente trabajo, titulado: Informatización del proceso de Revisiones Técnicas Formales en el área Arquitectura del desarrollo de software para el centro FORTES y autorizamos a la Universidad de las Ciencias Informáticas (UCI) para que haga el uso que estime pertinente con él.

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

Autores

Yairon Consuegra Cabrera

Gino Miguel Ricardo González

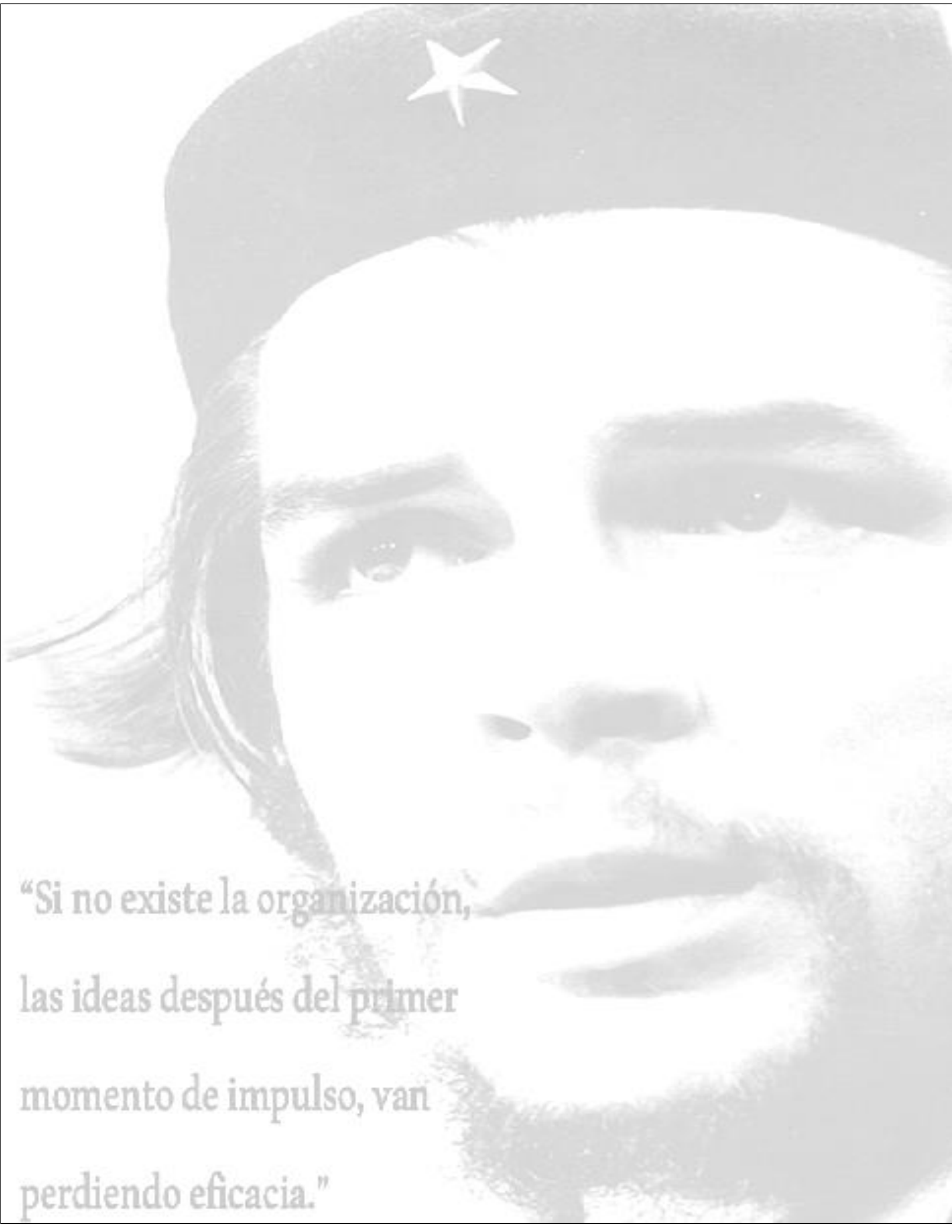
Tutores

Ing. Lisandra Peña Espinosa

Ing. Lizardo Ramírez Tabuada

Co-tutora

Ing. Roselia González Cardoso



“Si no existe la organización,
las ideas después del primer
momento de impulso, van
perdiendo eficacia.”

Agradecimientos

De Gíno:

A mi papá Yeyo y mi mamá María por ser únicos, y de gran ayuda en mi desarrollo profesional.

A Liuba por ser mi segunda madre, amiga, consejera y por brindarme sus conocimientos.

A mis abuelas Elvia y Carmen, mis primos, Lianne, Idelmer, por confiar en mí y darme su bendición para cumplir mis metas.

A mi tía Georgina y demás tíos por su apoyo incondicional.

A mis hermanos Jimmy y Glenda por darme alegría.

Agradecer a mis tutores Lizardo y Lizandra, a Leannys por el apoyo, su ayuda y confiar en mí para el desarrollo de este trabajo.

A Yairon por darme el voto de confianza para realizar este trabajo y por ser mi amigo.

A mi amigo y hermanito Abel por acompañarme en los estudios durante estos 5 años.

A Yunier Broche por su gran apoyo, por compartir sus conocimientos conmigo, gracias hermano.

A todos los que de una manera u otra han influido en mi superación personal y profesional.

De Yáiron:

*Agradezco a mis padres por estar siempre a mi lado, en las
malas y las buenas y por siempre confiar en mí.*

*A mis tutores Lisandra y Lizardo, por su paciencia y
dedicación.*

A Gino por siempre confiar en mí.

A Leannys por apoyarnos en todo momento.

A Broche por ayudarnos incondicionalmente.

A mi novia Lizaída por su gran amor, cariño y compañía

A mis amigos Maikel, Corchy por su compañía.

Dedicatoria

De Gíno:

A mi tía Dania por ser mi ejemplo a seguir en cuanto a superación y constancia se refiere, aunque no estés presente físicamente siempre te recordare , te quiero mucho por lo que significaste y por lo que sigues significando para mí.

De Yairon:

A mi mamá Ester y mi papá Tomás por ser mis ejemplos a seguir en cuanto a superación, confianza y constancia se refiere, les debo toda mi vida y lo que soy.

Resumen

La aplicación de las Tecnologías de la Información y las Comunicaciones ha permitido el desarrollo de sistemas que implementan acciones dirigidas a obtener el objetivo de disímiles entidades, haciendo necesaria su calidad. Una técnica para asegurar la calidad del software lo constituyen las Revisiones Técnicas Formales en áreas como la Arquitectura del desarrollo de software. En el caso del Centro de Tecnologías para la Formación (FORTES) de la Universidad de las Ciencias Informáticas todo el proceso se realiza de forma manual, dificultando el trabajo de los expertos, dando un margen para la ocurrencia de errores y frenando el logro de una arquitectura de proyecto totalmente definida. A estos elementos se une el empleo considerable de tiempo y el ineficiente seguimiento de los señalamientos resultantes. No existe un registro histórico de las evidencias de la evaluación impidiendo que los especialistas obtengan información sobre las evaluaciones realizadas. Se llevó a cabo un estudio de sistemas que informatizan dicho proceso concluyendo que estos se realizan mediante modelos o técnicas con fases específicas que no se adaptan a las revisiones definidas para los productos de FORTES. Por lo anteriormente expuesto se hace necesario informatizar el procedimiento de Revisiones Técnicas Formales en el área Arquitectura del desarrollo de software para el centro FORTES, siendo este el objetivo de la presente investigación. El proceso de desarrollo fue guiado por la metodología Extreme Programming y se implementó sobre los marcos de trabajo jQuery, Bootstrap, Symfony y Doctrine. Una vez concluida la investigación se puede afirmar que se cumplieron los objetivos trazados para el desarrollo del trabajo.

Palabras clave: Arquitectura del desarrollo de software, Extreme Programming, Symfony, Revisiones Técnicas Formales.

Índice

Declaración de autoría	I
Resumen	V
Introducción	1
Capítulo 1: Fundamentación teórica.....	5
Introducción.....	5
1.1 Conceptos asociados al dominio del problema	5
1.1.1 Calidad	5
1.1.2 Calidad de software	5
1.1.3 Arquitectura de software	6
1.1.4 Revisión o inspección formal de software	11
1.1.5 Revisiones Técnicas Formales orientadas a la Arquitectura de software	12
1.1.6 Listas de Chequeo.....	13
1.2 Sistemas informáticos estudiados.....	14
1.2.1 Sistema de colaboración a la revisión de software (CSRS, Collaborative software review system).....	14
1.2.2 Inspección de software colaborativa (CSI, Collaborative software inspection)	15
1.2.3 Inspección inteligente de código en ambiente de lenguaje C (ICICLE, Intelligent code inspection in a C language environment).....	16
1.2.4 Examen cuidadoso (Scrutiny)	16
1.2.5 Inspección de software en fases para asegurar la calidad (InspeQ, Inspecting software in phases to ensure quality).....	17
1.2.6 Valoración de los sistemas estudiados	17
1.3 Metodología de desarrollo del software.....	18
1.3.1 Metodologías ágiles.....	18
1.3.2 Metodologías tradicionales	19
1.4 Lenguaje de modelado	20
1.4.1 UML.....	20
1.5 Herramientas para el modelado.....	21
1.5.1 Visual Paradigm for UML	21
1.6 Lenguajes de desarrollo del software.....	21
1.6.1 Lenguajes del lado del cliente.....	22
Tecnologías del lado del cliente	23
1.6.2 Lenguajes del lado del servidor	23

1.7	Framework.....	24
1.7.1	Frameworks para la capa de presentación	24
1.7.2	Frameworks para la capa de lógica de negocio	25
1.7.3	Frameworks para la capa de acceso a datos	25
1.8	Sistema gestor de base de datos.....	26
1.8.1	PostgreSQL	26
1.9	Servidor web.....	26
1.9.1	Apache	27
1.10	Entorno de desarrollo.....	27
1.10.1	PhpStorm.....	27
1.10.2	NetBeans.....	27
	Conclusiones parciales del capítulo.....	28
	Capítulo 2: Diseño de la propuesta de solución	29
	Introducción.....	29
2.1	Flujo actual del procedimiento de las RTF en el área de Arquitectura.....	29
2.2	Propuesta de solución	30
2.3	Exploración	31
2.3.1	Historias de usuarios	31
2.3.2	Características del sistema	33
2.4	Planificación.....	34
2.4.1	Estimación de esfuerzo por historia de usuario.....	35
2.4.2	Plan de iteraciones	36
2.5	Diseño	38
2.5.1	Tarjetas CRC	38
2.5.2	Modelo de datos	40
2.5.3	Prototipos de interfaz de usuario.....	41
2.5.4	Patrón arquitectónico empleado.....	43
2.5.5	Patrones de diseño	46
	Conclusiones parciales del capítulo	50
	Capítulo 3: Implementación y pruebas de la propuesta.....	51
	Introducción	51
3.1	Mapa de navegación.....	51
3.2	Estándar de codificación	51
3.3	Implementación	52

3.3.1	Tareas de ingeniería	52
3.4	Pruebas	54
3.4.1	Pruebas de aceptación	54
3.4.2	Pruebas unitarias	57
	Conclusiones parciales del capítulo	59
	Conclusiones	60
	Recomendaciones	61
	Bibliografía	62
	Anexos	69

Introducción

En la contemporaneidad la aplicación de las Tecnologías de la Información y las Comunicaciones (TIC) se ha convertido en la base fundamental de la evolución y el progreso del hombre. Los novedosos avances de la informática como ciencia han influido de forma determinante en las diversas esferas de la sociedad.

Podría estimarse como considerable el número de empresas cuyo objeto social es desarrollar productos de software que puedan ser incorporados a los procesos fundamentales y a las estrategias administrativas de disímiles entidades. El empleo de sistemas de software implica no solo mejorar los procesos sino incrementar el desarrollo y los alcances. Dichos sistemas implementan acciones dirigidas a obtener el objetivo perseguido por los clientes, elemento que hace imprescindible emplear recursos en la obtención de soluciones con calidad; la cual se define como la capacidad del producto de software para permitirles a usuarios específicos lograr las metas propuestas con eficacia, productividad, seguridad y satisfacción en contextos especificados de uso (Lomprey, y otros, 2008).

Es imprescindible controlar la calidad durante todas las etapas del ciclo de vida del software. Se hace necesario el uso de metodologías y/o procedimientos estandarizados para el análisis, diseño, programación y prueba del software (Carrasco, y otros, 1995). Durante las fases se genera un conjunto de artefactos, dependientes de la metodología usada para guiar el desarrollo, cuya estructura, formato, estandarización, contenido e información influyen de forma determinante en el éxito del resultado final.

La Universidad de las Ciencias Informáticas (UCI) cuenta con centros tecnológicos como es el caso del Centro de Tecnologías para la Formación (FORTES) de la Facultad 4 especializado en la producción de aplicaciones y servicios informáticos orientados al sector educacional. Para asegurar la calidad de la documentación generada, los especialistas de dicho centro emplean las Revisiones Técnicas Formales (RTF), las cuales permiten reducir sustancialmente el costo del software además de generar un gran valor educativo para los participantes, como también fomentar la seguridad y continuidad del proyecto al que se le aplica (NU. CEPAL (Comisión Europea), 2009).

Las RTF validan la completitud y corrección de los entregables, previniendo en forma temprana defectos que puedan derivarse en etapas posteriores: inconsistencias, ambigüedades y no cumplimiento de estándares. Permiten obtener alertas tempranas sobre potenciales riesgos y problemas de calidad, reducir los tiempos de desarrollo y pruebas así como generar estándares útiles para el ciclo de desarrollo. Abarcan una revisión formal de la documentación de

arquitectura, diseño, requerimientos y modelo de datos con el objetivo de verificar la consistencia interna de la documentación y su coherencia con los requerimientos, el cumplimiento de estándares del cliente y la facilidad de lectura de la documentación, además de proponer mejoras, agregados, plantillas y/o estándares nuevos (Pressman, 2005).

En el centro FORTES existe un procedimiento que permite a los especialistas documentarse sobre cómo realizar las revisiones de software. En los proyectos pertenecientes al centro las actividades de control de la calidad se hacen durante todas las fases del ciclo de vida del software y sus datos son registrados en el Sistema de software para la Dirección Integrada de Proyectos (GESPRO) como herramienta adecuada para la gestión de proyectos. Las revisiones en el área de Arquitectura del desarrollo de software se efectúan empleando la Lista de Chequeo de Revisiones Técnicas Formales la cual cuenta con una serie de preguntas que verifican el cumplimiento satisfactorio de los entregables de la Arquitectura de Software, y permite evaluar los mismos.

A pesar de la existencia de este procedimiento no está definido formalmente cómo se debe revisar cada producto en el sistema de calidad establecido ni se cuenta con manuales digitales que apoyen la ejecución de esta actividad. Como resultado de las RTF se generan los informes técnicos, documentos que no son gestionados desde el GESPRO. Todo el proceso se realiza de forma manual, hecho que dificulta el trabajo de los expertos dando un margen para la ocurrencia de errores y frenando el logro de una arquitectura de proyecto totalmente definida. A estos elementos se une el empleo considerable de tiempo y el ineficiente seguimiento de los señalamientos resultantes de las revisiones. No existe un registro histórico de las evidencias de la evaluación impidiendo que los especialistas obtengan información sobre las evaluaciones realizadas con anterioridad.

Luego de analizar la problemática anterior, se plantea la siguiente interrogante como **problema a resolver**: ¿Cómo informatizar el procedimiento de Revisiones Técnicas Formales en el área Arquitectura del desarrollo de software para el centro FORTES?

Se determina como **objeto de estudio**: El procedimiento de Revisiones Técnicas Formales en el desarrollo del software.

Para dar solución al problema antes expuesto se formula el siguiente **objetivo general**: Informatizar el procedimiento de Revisiones Técnicas Formales en el área Arquitectura del desarrollo de software para el centro FORTES. Para dar cumplimiento al objetivo general, se definen como **objetivos específicos**:

- Investigar el proceso de Revisiones Técnicas Formales, sus conceptos fundamentales y componentes o artefactos que se analizan en la Arquitectura del desarrollo de software de manera general y en el centro FORTES.
- Diseñar una herramienta para la Revisión Técnica Formal en el área de Arquitectura del desarrollo de software.
- Realizar la implementación y pruebas de la herramienta para las Revisiones Técnicas Formales en el área de Arquitectura del desarrollo de software.

El **campo de acción** lo constituye la informatización del procedimiento de las Revisiones Técnicas Formales en el área de Arquitectura del desarrollo de software en el centro FORTES. Con vista a resolver el problema planteado se propone la siguiente **hipótesis**: La informatización del procedimiento en el área de Arquitectura del desarrollo de software permitirá agilizar las Revisiones Técnicas Formales en el centro FORTES.

Tareas de investigación

- Identificación y análisis de las tendencias actuales sobre las revisiones técnicas formales en el área de Arquitectura del desarrollo de software.
- Estudio y selección de la metodología empleada para llevar a cabo el desarrollo de la herramienta.
- Selección de las tecnologías informáticas a utilizar en el desarrollo de la herramienta.
- Descripción de las funcionalidades y características de la herramienta.
- Diseño de la herramienta utilizando la metodología seleccionada.
- Implementación y validación de la herramienta.

En la investigación fueron usados diferentes **métodos científicos** con el fin de darle solución a las tareas de investigación. A nivel teórico fue utilizado:

El **histórico-lógico** con el propósito de realizar un estudio teórico desde una lógica adecuada sobre las revisiones técnicas formales en el área de arquitectura y los elementos que inciden en su realización. Además, conocer la evolución de las diferentes herramientas que realizan inspecciones de software o revisiones técnicas, evidenciándose como antecedentes a la nueva solución propuesta.

El **analítico-sintético** que brindó la posibilidad de realizar un estudio bibliográfico profundo de la teoría existente alrededor del objeto de estudio, determinar las características que tendrá la propuesta de solución y definir las tecnologías y herramientas más adecuadas para el desarrollo de la propuesta de solución.

El **inductivo-deductivo** que permitió deducir características aplicables al nuevo sistema como parte de su integración, después de haber realizado un análisis inductivo de los artefactos previos.

Fueron utilizados a nivel empírico los siguientes métodos:

La **entrevista** que permitió realizar conversaciones con especialistas en revisiones técnicas formales en el área de arquitectura y con el arquitecto del centro FORTES para obtener información necesaria para la identificación y especificación de los requisitos. El tipo de entrevista realizada fue la informal; no se definieron fechas para la misma y las preguntas no fueron planificadas, sino que se desarrollaron de acuerdo a la fluidez de la conversación. Fue una entrevista no estructurada pues se trabajó con preguntas abiertas, sin un orden preestablecido, que se hicieron de acuerdo a las respuestas que fueron surgiendo durante la entrevista. Fueron consultados los siguientes especialistas:

Yosvany Márquez Ruiz, arquitecto de calidad de software de la UCI.

Luz María Gutiérrez Fera, especialista de la dirección de calidad de software de la UCI.

Ernesto Vladimir Pereda, arquitecto de la Facultad 4.

La **observación** se utilizó en diferentes momentos de la investigación para agrupar elementos relacionados con la problemática en cuestión y para identificar de los sistemas similares las mejores prácticas, así como las vulnerabilidades.

El presente trabajo consta de tres capítulos desarrollados a partir del estudio realizado.

En el **capítulo 1**: se hace un estudio bibliográfico sobre los elementos teóricos que aplicarán en la investigación. Además se realiza un análisis sobre los sistemas que realizan revisiones técnicas formales en el área de arquitectura. Se valoran e identifican las metodologías, tecnologías y herramientas utilizadas en el desarrollo de software.

En el **capítulo 2**: se realiza el diseño de la propuesta de solución de la presente investigación con el cumplimiento de las actividades propuestas por las fases de la metodología seleccionada para guiar el proceso de desarrollo de la solución. Se identificaron los requisitos funcionales y no funcionales y se muestran los artefactos generados.

En el **capítulo 3**: se presentan los aspectos fundamentales de la fase de construcción, dentro de la que destacan los procesos de implementación y prueba de software. Se muestran los resultados obtenidos y los casos de prueba para validar la hipótesis base de la investigación.

Capítulo 1: Fundamentación teórica

Introducción

En el presente capítulo se realiza un estudio del estado del arte relacionado con el tema que ocupa la presente investigación. Se analizan las ventajas, desventajas y las características esenciales que distinguen las herramientas y metodologías más usadas en el desarrollo de sistemas para la informatización de las revisiones técnicas formales en el área de Arquitectura del desarrollo de software para definir cuál será utilizada.

1.1 Conceptos asociados al dominio del problema

1.1.1 Calidad

Rafael Picolo, director general de Hewlett Packard, afirma que la calidad no se puede tratar como un concepto aislado. Descansa en fuertes valores que se presentan en el medio ambiente así como en otros que se adquieren con esfuerzo y disciplina (Picolo, 2006).

Según la norma ISO 8402 la calidad es la totalidad de las características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas (Acuña, 2001).

Por otra parte el Diccionario de la Real Academia Española la define como la propiedad o conjunto de propiedades inherentes a algo que permiten juzgar su valor; es sinónimo de buena calidad la superioridad o excelencia (Alfaro, 2008).

Se concluye que la calidad resume las características, propiedades, cualidades y atributos propios de un producto que determina sobre este la ausencia de defectos y la conformidad de todo el personal vinculada a él.

1.1.2 Calidad de software

El concepto de calidad de software ha evolucionado considerablemente con el paso de los años y los avances alcanzados en el desarrollo de aplicaciones.

El Instituto de Ingenieros Eléctricos y Electrónicos la define como la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente (Álvarez, 2008).

En el artículo titulado *Gestión, control y garantía de la calidad del software* se plantea que es la ausencia de errores de funcionamiento, la adecuación a las necesidades del usuario y el alcance de un desempeño adecuado, además del cumplimiento de los estándares (Antonio, 2008).

Después de consultar las definiciones anteriores es posible afirmar que la calidad de software es un conjunto de cualidades que lo caracterizan y determinan su utilidad y existencia; está dada por las necesidades del cliente. Además se plantea un balance adecuado de eficiencia, confiabilidad, facilidad de mantenimiento, portabilidad, facilidad de uso, seguridad e integridad.

1.1.3 Arquitectura de software

Es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores y analistas compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software. (Osvaldo Díaz Verdecia, 2009).

Clements la define como una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión del detalle inherente a la mayor parte de las abstracciones (Reinoso, 2014).

Teniendo en cuenta estas definiciones de arquitectura de software se puede señalar que es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software. Establecen la estructura, funcionamiento e interacción entre las partes del software así como las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema.

Las vistas arquitectónicas constituyen un elemento imprescindible de la arquitectura de software por lo que se hace necesario precisar su definición así como los principales elementos que forman parte de ellas (Lazo, 2011).

Vistas arquitectónicas

Las vistas arquitectónicas componen las estructuras esenciales de la arquitectura de software y hacen posible enfrentar su diseño reduciendo la complejidad presente en su desarrollo. El diseño de las vistas arquitectónicas implica establecer una sincronización entre sus elementos para conformar una arquitectura integrada.

Las vistas arquitectónicas han sido identificadas y descritas por un número considerable de autores; la presente investigación asumirá las definidas por el modelo de las 9+1 vistas desarrollado en la UCI, en el cual se propone un modelo arquitectónico de referencia donde se integran las corrientes más representativas, centrándose en el papel de este proceso en la gestión de los proyectos informáticos, la definición de los productos y su evolución.

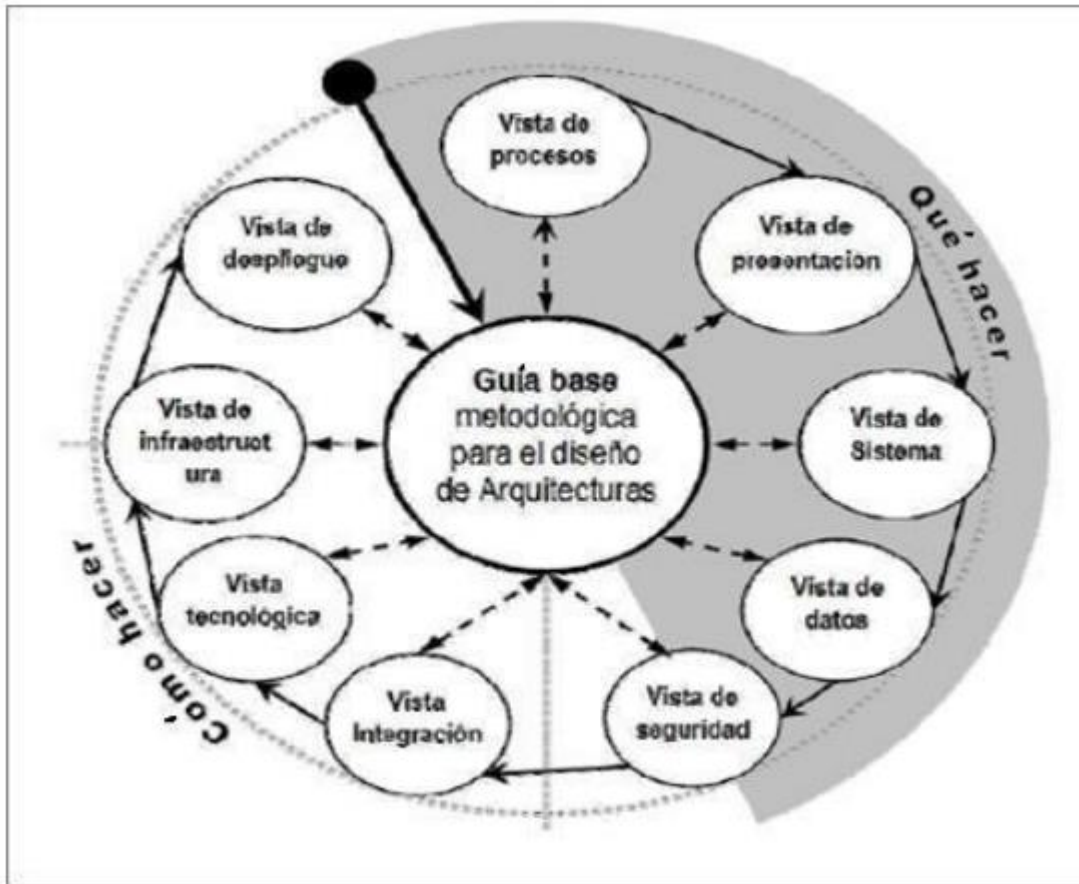


Figura 1: Guía base de análisis arquitectónico.

Vista de procesos

Es una abstracción o artefacto que da lugar a la discusión del marco estructural de la arquitectura del negocio. Para su formalización es necesario la definición de los macro procesos de una organización, así como el dominio (líneas temáticas), los procesos (son la base del centro, o sea, lo que se va a hacer), las normas (estándares) y las reglas del negocio. El desarrollo de la misma se basa en cuatro pasos fundamentales definidos en la Tesis para optar por el grado de Máster en Ciencias Técnicas del Máster René Lazo Ochoa:

1. **Caracterización del entorno según la vista.** Este primer paso es para identificar el dominio al que pertenece la solución propuesta. Dichos dominios están definidos en la planilla de arquitectura de la Vista de Procesos.
2. **Conceptualizar los macro procesos del entorno corporativo.** La función de este paso es identificar las áreas de procesos y procesos concretos, clasificándolos en claves, estratégicos o de soporte.

3. **Diseñar un mapa conceptual por procesos.** Este paso tiene como objetivo principal diseñar un mapa conceptual por cada uno de los procesos, caracterizando cada concepto en cuanto a tipo de datos, composición y excepciones, basándose en la identificación y caracterización de los estándares de información y conceptos establecidos, así como las reglas de negocio. Se debe además, describir los macro procesos de la organización que tengan impacto en la arquitectura. Estos procesos son aquellos que contienen las funcionalidades fundamentales que se desean informatizar.
4. **Priorización de los procesos del negocio.** Este paso consiste en priorizar los procesos teniendo en cuenta su relevancia para el negocio y su impacto para la arquitectura. Los procesos de mayor prioridad deberán ser considerados para las pruebas de concepto de arquitectura y ser analizados en detalle en la vista de sistema (Lazo, 2011).

Vista de presentación

Una vista de presentación es una descripción lógica de una funcionalidad del sistema que expresa una secuencia de intercambios de mensajes entre la parte funcional y los actores del sistema.

Representa además un área de la interfaz gráfica de usuario, que contiene aquellos elementos gráficos que responden a dicha porción de funcionalidad. En esencia describe cómo luce el software, cuáles son los colores que lleva la aplicación, cómo son los botones, los links, etc.

En la vista de presentación se deben de tener en cuenta elementos importantes como por ejemplo: la definición de pantallas, que contempla tres tipos fundamentales (Verdecia, y otros, 2010):

- Pantalla de tipo genérica.
- Pantalla de bienvenida.
- Pantalla de acceso a módulos.

Vista de sistema

La arquitectura de sistema es responsable de definir correctamente cohesionados, acoplados e interrelacionados los elementos computacionales del producto, las principales interacciones, los conectores y las configuraciones a asumir. La vista de sistema de la arquitectura de software representa una proyección simétrica de alto nivel, de los procesos de negocio o arquitectura de negocio que se trabaja, expresada en elementos, conectores, configuraciones y restricciones.

La vista de sistema en el modelo planteado por la Universidad de las Ciencias Informáticas contempla aspectos relacionados con la personalización del producto según la vista, donde se realiza una caracterización atendiendo a los elementos: personal de desarrollo, dinamismo,

cultura, tamaño del equipo y criticidad. Otros aspectos que se definen son: la representación de la dependencia entre paquetes, la descripción de los escenarios y patrones de solución, el agrupamiento en subsistemas o paquetes, la priorización de los requisitos, la identificación de los paquetes y componentes principales y la representación de la dependencia entre paquetes tanto en paralelo como en sucesión (Verdecia, y otros, 2010).

Vista de datos

La arquitectura de datos, identifica y precisa las mejores clases de datos que apoyan las funciones del negocio definidas en el modelo de negocios. Es la primera de las arquitecturas a ser concretadas porque la calidad de los datos es el producto básico de la función de la ingeniería de software. Tiene como objetivo puntualizar los principales tipos y fuentes de datos necesarios para dar soporte a las actividades de la empresa, de manera que sean entendibles por los participantes, estables, completas y consistentes.

Esta vista especifica arquitectónicamente los elementos constantes en el modelo de datos. Describe una apreciación global del mismo y su organización. También describe la cartografía de clases constantes de la vista lógica a la estructura de los datos de la base de datos. Además se desarrolla un diccionario de datos donde se relacionan todos los elementos necesarios para identificar los conceptos que influyen significativamente en la modelación de la base de datos y por último se construye el modelo de datos que recoge las especificaciones del diseño lógico y físico de la solución, es decir de la base de datos (Verdecia, y otros, 2010).

Vista de seguridad

La arquitectura de seguridad es la encargada de establecer toda la política de seguridad a seguir en las diferentes fases o entornos de un sistema.

En el modelo planteado por la UCI se incluye la vista de seguridad y esta contempla las siguientes especificaciones de seguridad:

- Datos: donde se define la seguridad de los datos y el acceso a los mismos.
- Código: donde se especifica el estado del código (ofuscado y sellado, así como las tecnologías usadas para esto).
- Nodos de despliegue e integración.
- Capa de presentación y entorno de trabajo (Verdecia, y otros, 2010).

Vista de integración

La arquitectura de integración pretende obtener una forma más eficiente y flexible de combinar recursos, con el objetivo de optimizar operaciones a través y más allá del medio ambiente organizacional. Provee una vista única consolidada a partir de conectores que definen y especifican el comportamiento e interacción entre elementos del negocio (sistemas,

subsistemas y componentes). Se analizan los procesos de integración del negocio; se identifican los principales flujos de colaboración o unificación en la arquitectura; se establecen los conceptos más críticos en la integración de sus procesos, de acuerdo al nivel de incidencia en los mismos, partiendo del análisis de su arquitectura, lo que permite conocer las áreas más críticas de integración y por último se analizan entradas y salidas de cada componente identificado por la arquitectura de sistema. Esta actividad permite estar al tanto de los lazos de colaboración entre componentes y clasificarlos según la estrategia de integración identificada por el grupo de arquitectura, construyéndose de esta manera la matriz de integración del negocio.

Esta vista se encarga de la selección de los estilos arquitectónicos a emplear, también está presente la selección de la estrategia de diseño y modelo de desarrollo a utilizar, así como la selección de los principales patrones de diseño, vista desde la integración entre las soluciones y la selección de los estándares de codificación (Verdecia, y otros, 2010).

Vista tecnológica

La vista tecnológica tiene como objetivo especificar y describir las tareas y competencias de los roles así como las actividades y artefactos del área de tecnología, y por tanto, definir las habilidades y los conocimientos que deben desarrollar las personas que trabajan en esta área de la arquitectura de software para lograr un buen desempeño. Es la responsable de garantizar un soporte tecnológico para el desarrollo de las configuraciones que propone el resultado de una arquitectura de sistema, así como las bases tecnológicas para los *frameworks* especializados de la arquitectura de sistema o de otras áreas como la de integración. Esta es la vista responsable de identificar las tecnologías y herramientas a usar en la realización de la aplicación, además de definir la factibilidad técnica del producto. Los integrantes de la vista de tecnología son los responsables de generar una tecnología tipo para la aplicación que se desea desarrollar, de transmitir el conocimiento al resto del proyecto y por último implantar y controlar el uso correcto de la tecnología, ganando autoridad y prestigio dentro del proyecto (Verdecia, y otros, 2010).

Vista de infraestructura

La vista de infraestructura se encarga de describir todo lo relacionado con las redes y las conexiones, así como las aplicaciones legadas e interrelacionadas. Cuenta con un grupo de componentes de suma importancia, entre los que se encuentran tres diagramas principales a través de los cuales se define esta vista: Diagrama de Redes, Diagrama de Software y Diagrama de Hardware. Estos a su vez contienen elementos de vital importancia entre los que se encuentran: redes, velocidad, protocolos, servidores, servicios y plataformas.

La vista de infraestructura en el modelo realizado por la UCI plantea cuatro aspectos fundamentales para describir y definir la vista.

1. Describir el diagrama de configuración de redes para el cual está previsto el despliegue de la solución.
2. Describir el diagrama de configuración del software.
3. Describir el diagrama de configuración de hardware.
4. Describir los escenarios y patrones de solución para la vista de infraestructura (Verdecia, y otros, 2010).

Vista de despliegue

Esta vista es la encargada de contemplar todo lo relacionado con el lugar donde se va a instalar el sistema en cuestión. Cuenta con los siguientes aspectos a desarrollar:

- Caracterización de la solución vista desde las licencias de los componentes que la forman: Donde se seleccionan los tipos de licencias que soportan los componentes de la solución.
- Caracterización de la solución vista desde las estrategias de desarrollo empleadas para su elaboración, donde se seleccionan las estrategias de desarrollo empleadas.
- Definición del tipo de licenciamiento de la solución vista desde la arquitectura: Donde se selecciona la estrategia de licenciamiento a utilizar.
- Definición de las estrategias de ingreso vista desde la arquitectura: Donde se selecciona la estrategia para la generación de ingresos.
- Definición de la estrategia de empaquetamiento del producto para la implantación.
- Declaración de los requerimientos mínimos de instalación.
- Declaración de la estrategia para el soporte y los detalles de los niveles de soporte que se van a realizar (Verdecia, y otros, 2010).

1.1.4 Revisión o inspección formal de software

Para determinar el concepto de revisión formal de software se hace necesario el estudio de las definiciones propuestas por diversos autores.

Según lo planteado por un prestigioso colectivo de autores de la Facultad de Informática, Ciencias de la Comunicación y Técnicas Especiales de la Universidad de Morón en su artículo titulado *Calidad de software* consiste en la detección de defectos, eliminación, verificación y corrección de un conjunto de requisitos realizada por un pequeño grupo de especialistas durante el ciclo de vida de desarrollo. Un defecto es cualquier error, no conformidad o incumplimiento de un requisito del producto. El objetivo de las inspecciones formales es

asegurar que los defectos se fijen al principio del ciclo de vida, pues posteriormente son más difíciles de encontrar y más costosos de arreglar (Universidad de Morón, 2011).

Por otra parte Pressman refiere que es el filtro más efectivo desde el punto de vista de aseguramiento de la calidad. Constituye un medio efectivo para descubrir errores durante el proceso de modo que no se conviertan en defectos después de liberar el software. Se llevan a cabo por individuos de desarrollo, prueba y aseguramiento y puede incluir a los usuarios (Pressman, 2005).

A partir de las definiciones estudiadas se puede concluir que las revisiones formales de software son el filtro más efectivo desde el punto de vista de aseguramiento de la calidad. Permiten obtener alertas tempranas sobre potenciales riesgos y problemas de calidad, reducir los tiempos de desarrollo y pruebas así como generar estándares útiles para los distintos ciclos de desarrollo. Abarcan una revisión formal de la documentación de arquitectura, diseño, requerimientos y modelo de datos con el objetivo de verificar la consistencia interna de la documentación y su coherencia con los requerimientos, el cumplimiento de estándares del cliente y la facilidad de lectura de la documentación, además de proponer mejoras, agregados, plantillas y/o estándares nuevos.

1.1.5 Revisiones Técnicas Formales orientadas a la Arquitectura de software

Una vez realizado el estudio de conceptos asociados a la arquitectura de software y las RTF es posible referir que:

Durante la fase de Arquitectura, se describen los componentes del sistema, su interfaz y cómo interactúan entre ellos. Se establece el esqueleto sobre el cual se implementarán los casos de uso en los equipos de desarrollo. El diseño arquitectónico debe ser inspeccionado por la satisfacción y trazabilidad de los requisitos, corrección, claridad, codificabilidad, capacidad de prueba y coherencia. Con el objetivo de evaluar el estado de la arquitectura del proyecto, detectar errores que pueden repercutir en la implementación, lograr que se alcancen los requerimientos y proporcionar mejoras, se debe realizar RTF a la documentación asociada al expediente de arquitectura.

Pasos para revisar la Arquitectura de un proyecto

- Solicitar la documentación asociada al expediente de arquitectura.
- Revisar la documentación y elaborar un resumen.
- Realizar una entrevista al grupo de arquitectos.
- Revisar las evidencias asociadas a cada vista de la arquitectura.
- Emitir el dictamen sobre el estado de elaboración del artefacto: Diseño de la Arquitectura de Software.

Pasos genéricos de cada una de las vistas

El chequeo de la arquitectura de un proyecto se basa fundamentalmente en el análisis de los elementos más significativos de las vistas arquitectónicas, derivado de un conjunto de pasos genéricos presentes en todas las vistas:

- Caracterización del producto o el entorno según la vista, puede incluir varias actividades de análisis específicos y la alineación de la vista con los requisitos y las restricciones del producto o el dominio.
- Análisis y descripción de los escenarios y patrones de solución si procede. Toma de decisiones arquitectónicas. Análisis de las metas arquitectónicas y su impacto en los atributos de calidad.
- Construcción de artefactos descriptivos de acuerdo a la metodología de desarrollo empleada como los gráficos, diagramas y esquemas.
- Pruebas de conceptos sobre las decisiones tomadas. Análisis de los principales indicadores de calidad de la solución técnica abstraída (Ruiz, 2014).

En cada vista se toman decisiones arquitectónicas que inciden en los diferentes atributos de calidad de la arquitectura.

1.1.6 Listas de Chequeo

Las Listas de Control u Hojas de Verificación, son formatos creados para realizar actividades repetitivas, controlar el cumplimiento de una lista de requisitos o recolectar datos ordenadamente y de forma sistemática, asegurándose que el trabajador o inspector no se olvida de ningún elemento de importancia. Es un listado de preguntas en forma de cuestionario que sirve para verificar determinadas reglas establecidas a priori con un fin determinado (Casal, 1989).

Según lo planteado por Fabián Oliva es una herramienta que utiliza preguntas orientadas a identificar problemas por las diversas áreas de un proyecto y se utilizan para motivar posibles soluciones o detectar oportunidades de mejora (Mella, 2009).

Las listas de chequeo o de verificación determinan el grado de peligrosidad que se tiene en un proyecto, de tal forma que se pueda tomar las medidas necesarias, para tratar de eliminar o en caso contrario reducir ese peligro de modo que no rompa con el continuo desarrollo de las actividades programadas (Bichachi, 2010).

Los principales usos de las listas de chequeo son los siguientes:

- Realización de actividades en las que es importante que no se olvide ningún paso y/o deben hacerse las tareas con un orden establecido.

- Realización de inspecciones donde se deja constancia de cuales han sido los puntos inspeccionados (Casal, 1989).
- Verificar o analizar artículos, operaciones y defectos.
- Identificar las causas de los defectos.
- Recopilar datos para su futuro análisis.

Aspectos que conforman la lista de chequeo para evaluar el estado de la Arquitectura
(Mella, 2009)

- Funcionalidad
- Estudios comerciales
- Rendimiento
- Lógica
- Uso de datos
- Interfaces
- Capacidad de prueba

Luego de analizar los conceptos anteriores se concluye que las listas de chequeo son un conjunto de preguntas, con un orden establecido, en forma de cuestionario que asegura el cumplimiento de determinadas actividades en las diferentes áreas del proceso de desarrollo de un software. Se detectan errores y se proponen mejoras con anterioridad para evitar errores futuros. Proporciona eficiencia y calidad a la documentación y permite que al final del proyecto o durante el desarrollo del mismo se realicen evaluaciones, revisiones y auditorías.

1.2 Sistemas informáticos estudiados

En el mundo existen diversos sistemas que informatizan los procesos relacionados con las RTF. Algunos de ellos son:

1.2.1 Sistema de colaboración a la revisión de software (CSRS, Collaborative software review system)

El sistema de colaboración a la revisión de *software* (CSRS) es un ambiente para apoyar el uso de FTArm (método asincrónico técnico formal) de la revisión, un desarrollo obtenido a partir del método de inspección de Fagan¹ (Sotés, 2012).

¹

El método de inspección de Fagan se refiere al proceso estructurado de intentar encontrar defectos en documentos de desarrollo tales como código de programación, especificaciones y diseño, durante las fases del proceso de desarrollo del software. Es llamado así debido a Michael Fagan, quien es reconocido como el inventor de la inspección formal del software.

Entre sus características destacan:

- Es un método general para examinar cualquier tipo de documento.
- Un documento se almacena en una base de datos como una serie de nodos interrelacionados.
- Las anotaciones también se almacenan como nodos de tres tipos: nodo del comentario, de la edición y acción.
- Proporciona la colección automática de los datos tales como número y severidad de los defectos.
- También tiene la capacidad de guardar un registro de los acontecimientos.
- Proporciona una lista de comprobación en línea para asistir al inspector.

1.2.2 Inspección de software colaborativa (CSI, Collaborative software inspection)

Está diseñado para apoyar la inspección de todos los productos del desarrollo del software. La herramienta se describe en relación al modelo de inspección de Humphrey². En esta variación, cada inspector crea una lista de los defectos durante la inspección individual que se hace llegar al autor del documento. Es la tarea del autor correlacionar estas listas individuales y tratar cada defecto.

Entre sus características destacan:

- Proporciona un navegador para ver el documento bajo inspección que numera automáticamente cada línea.
- Puede anotarse qué es lo que falta en el artefacto.
- El inspector puede categorizar y clasificar los defectos.
- Presenta un resumen al autor clasificando cada defecto y este puede aceptar o rechazarlo.
- Se puede tener un acceso a todas las anotaciones asociadas al documento y correlacionarlas en una sola lista de defectos.
- Permite al autor clasificar la lista de defectos en opciones múltiples, incluyendo severidad, la época de la creación y el estado actual.
- Prevé la inspección distribuida, permitiendo una inspección con los miembros del equipo en una variedad de lugares.
- Proporciona un registro de la historia de la inspección (Sotés, 2012).

² El método de inspección de Humphrey es el conjunto de inspecciones dirigidas a los procesos, metodologías, planes o a cualquier artefacto producido en el transcurso del desarrollo de software, detectando algún defecto en estos.

1.2.3 Inspección inteligente de código en ambiente de lenguaje C (ICICLE, Intelligent code inspection in a C language environment)

ICICLE, el nombre sugiere que es un ayudante inteligente automatizado para la inspección del código de C. Esta herramienta de inspección es única y hace uso de conocimiento para asistir a encontrar defectos comunes. El conocimiento está en una clase muy específica, que controla la base del lenguaje C, por lo que ICICLE no es conveniente para apoyar a una inspección general.

Entre sus características destacan (Sotés, 2012):

- El código fuente se exhibe en una ventana con cada línea numerada.
- Prepara automáticamente comentarios sobre código fuente usando sus herramientas de análisis.
- Se puede utilizar para señalar defectos en una línea específica del código fuente.
- Capacidad de modificar e incluir procedimientos para requisitos particulares del análisis.
- Proporciona la información de los objetos tales como variables y funciones en C.
- Genera una lista de todos los defectos encontrados.
- Genera un resumen de los defectos por tipo, clase y severidad.

1.2.4 Examen cuidadoso (Scrutiny)

La herramienta Scrutiny permite realizar una inspección en general de los productos en su ciclo de vida. Permite la inspección distribuida, puede ser integrada con las herramientas estudiadas y adaptada para apoyar diversos procesos del desarrollo. La implementación de sus funcionalidades se encuentran basada en roles.

Entre sus características destacan (Sotés, 2012):

- Existe una ventana donde se puede visualizar el documento que está en inspección.
- Apoya, solamente, a documentos de texto.
- No utiliza la lista de comprobación ni otra documentación de soporte.
- Todos los defectos se encuentran manualmente.
- Se ha probado con inspecciones locales y distribuidas.
- Se ha utilizado instalaciones de audio y de tele conferencia.
- Un inspector puede enviar puntos de discusión al resto de los participantes.
- Proporciona medios para enviar mensajes simples a los participantes de la revisión.
- Genera automáticamente un informe que contiene toda la información relevante sobre la inspección y sus participantes.

- Obtiene una lista completa de defectos con la información necesaria.

1.2.5 Inspección de software en fases para asegurar la calidad (InspeQ, Inspecting software in phases to ensure quality)

InspeQ, es un conjunto de herramientas desarrollado por Knight y Myers, para apoyar su técnica propuesta para la inspección. La técnica fue desarrollada por Knight y Myers con la meta de permitir que el proceso de la inspección sea riguroso, justo, eficiente en su uso de recursos y apoyada por la computadora. A pesar de esto, entre los sistemas existentes en la actualidad, el InspeQ está considerado como uno de los que menos soporte brinda a la gestión de documentos. Posee un desempeño favorable en la etapa de la preparación individual asistiendo al trabajo del inspector mediante el uso de listas de comprobación y asegurándose que cada punto de la lista es vista por el inspector. Permite además el uso de documentos que representen estándares.

Entre sus características destacan (Sotés, 2012):

- Existen dos tipos de fases: solo un inspector y múltiples inspectores.
- Se utiliza una lista de comprobación rigurosa.
- Las inspecciones en fase se diseñan para permitir que los expertos se concentren en encontrar defectos.
- El inspector puede examinar simultáneamente partes separadas del mismo documento.
- InspeQ realiza el formato de los comentarios antes de que se pasen al autor.
- Asegura que todos los artículos de la lista de comprobación son tratados por el inspector antes de las salidas del producto y la fase.
- Apoya la supervisión de personal.
- Genera una lista de comentarios para cada inspector.

1.2.6 Valoración de los sistemas estudiados

Se pueden ver las siguientes características en los sistemas de software estudiados:

- Todas las herramientas describen los documentos de texto adecuadamente.
- ICICLE, Scrutiny y CSI utilizan la misma técnica de exhibir el documento con cada línea numerada.
- CSRS divide el documento en pedazos más pequeños llamados nodos.
- InspeQ tiene los comentarios fuera del documento fuente.
- ICICLE, CSI, Scrutiny y CSRS realizan la clasificación de los defectos y sus anotaciones, mientras que InspeQ permite solamente su creación o cancelación.
- Las listas de comprobación son apoyadas solamente por InspeQ.

- CSI tiene el concepto de una lista de los criterios que ayude a los inspectores a encontrar y categorizar los defectos.
- InspeQ exhibe estándares, mientras que ICICLE puede proporcionar la facilidad de hojear las páginas de los manuales dentro del software.
- ICICLE es la única herramienta para proporcionar cualquier detección automática de defecto para código en C.
- CSRS puede proporcionar los detalles de la cantidad de tiempo que pasa un inspector al momento de hacer la inspección.
- ICICLE recolecta automáticamente métricas en el número y tipo de comentarios hechos, así como su severidad.
- CSI utiliza un registro para la métrica del defecto incluyendo la severidad, tiempo tomado para encontrar el defecto e intervalo de tiempo total.

Una vez analizadas las capacidades de los sistemas de software estudiados se puede concluir que informatizan parte de los procesos de revisiones guiados por diversos modelos o técnicas (Fagan, Humphre y técnica de Knight y Myers para la inspección) que poseen fases bien específicas que no se adaptan a las revisiones definidas para los productos del centro FORTES.

De esta forma se pone de manifiesto la necesidad de implementar una solución que responda adecuadamente a las RTF que se llevan a cabo en el centro FORTES. Terminado este estudio se provee a la investigación de un punto de partida para la creación de la solución.

1.3 Metodología de desarrollo del software

Una metodología de desarrollo de software hace posible la planificación, organización y construcción de un sistema o proyecto con independencia de su temática o complejidad. Actualmente es una guía en el proceso de desarrollo de las aplicaciones informáticas, permitiendo que se obtengan resultados con la mayor calidad, rapidez y eficiencia posible, para evitar cometer errores futuros (Pressman, 2002).

Para seleccionar qué metodología es la más adecuada en el desarrollo del proceso de construcción de software en la presente investigación, se realizó un estudio de las metodologías ágiles y tradicionales.

1.3.1 Metodologías ágiles

Están basadas en una fuerte interacción con el cliente y usuarios que permite obtener productos adecuados a las necesidades reales, ahorrando esfuerzo y aumentando la satisfacción del usuario final. Ofrecen solución, casi a la medida, para gran cantidad de proyectos pequeños y con requisitos muy cambiantes. Se caracterizan por su sencillez, tanto en

el aprendizaje como en su aplicación, reduciendo los costos de implantación en un equipo de desarrollo. Son especialmente orientadas para proyectos pequeños, principalmente para aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías (Canós, y otros, 2003).

1.3.2 Metodologías tradicionales

Las metodologías de desarrollo robustas o tradicionales son usadas para desarrollar soluciones empresariales complejas. Guían a los equipos de proyecto sobre cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos del negocio, el tiempo al mercado y los riesgos. Centrar su atención en llevar una documentación exhaustiva y en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo. Otra de las características importantes dentro de este enfoque es los altos costos al implementar un cambio al no ofrecer una buena solución para proyectos donde el entorno es volátil (Shirley Fabiola, 2012).

Luego de realizado el estudio anterior, los autores del presente trabajo seleccionaron las metodologías ágiles como las que más se adaptan al desarrollo de la propuesta de solución de la investigación teniendo en cuenta la existencia de un equipo de desarrollo pequeño y requisitos cambiantes. A continuación se selecciona, dentro de las metodologías ágiles, cuál es la más adecuada para ser utilizada en el proceso de desarrollo. Para ello se muestra un estudio de las principales metodologías de esta clasificación.

SCRUM: indicada para proyectos con cambios considerables de requisitos. Se basa en la adaptación continua a las circunstancias de la evolución del proyecto. El desarrollo de software se realiza mediante iteraciones, con una duración de 30 días. El resultado de cada iteración es un incremento ejecutable que se chequea con el cliente. Consta de reuniones a lo largo del proyecto donde el equipo de desarrollo coordina e integra todo el proceso productivo. Controla de forma empírica y adaptable la evolución del proyecto empleando las siguientes prácticas de la gestión ágil (Palacio, 2006):

- Revisión de las iteraciones
- Desarrollo incremental
- Desarrollo evolutivo
- Auto-organización
- Colaboración

Crystal Methodologies: conjunto de metodologías para el desarrollo del software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El equipo de desarrollo es un factor clave, por lo

que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo bien definidas. El desarrollo de software se considera un juego cooperativo de invención y comunicación limitado por los recursos a utilizar (Abreu, y otros, 2008).

Proceso Unificado Ágil (AUP): es una forma simplificada del RUP. Describe un enfoque simple del desarrollo del software usando técnicas y conceptos ágiles. Algunas técnicas usadas por AUP incluyen el desarrollo orientado a pruebas, modelado, gestión de cambios ágiles y refactorización de base de datos para mejorar la productividad. Abarca siete flujos de trabajo, cuatro ingenieriles y tres de apoyo: Modelado, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyectos y Ambiente. Dispone de cuatro fases igual que RUP: Incepción o Creación, Elaboración, Construcción y Transición (Donatien, 2011).

Extreme Programming (XP): centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo y preocupándose por el aprendizaje de los desarrolladores. Se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo. Presenta características como la comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas. Se define adecuada para proyectos con requisitos imprecisos y muy cambiantes donde existe un alto riesgo técnico (Letelier, y otros, 2013).

Metodología seleccionada

Para determinar la metodología más adecuada para guiar la propuesta de solución se definieron algunas condiciones existentes que determinaron la selección de XP: poco personal para llevar a cabo el desarrollo, poco tiempo para el desarrollo, requisitos cambiantes a lo largo de todo el ciclo de vida del software y presencia del cliente dentro del proceso.

Cabe destacar que la curva de aprendizaje de XP es rápida, pero lleva tiempo acostumbrarse a estas técnicas. Sin embargo, los autores del presente trabajo no tienen tal desventaja dado que ya poseen conocimiento de las prácticas básicas de XP, así como cierto grado de experiencia.

1.4 Lenguaje de modelado

1.4.1 UML

Lenguaje Unificado de Modelado (LUM o UML, por sus siglas en inglés, Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar un sistema. Dispone un conjunto de notaciones y diagramas estándares para modelar sistemas orientados a objetos, describiendo la semántica esencial de lo que estos significan (Umbrello UML Modeller Autores, 2003).

Implementa un lenguaje de modelado común para todos los programadores mediante una documentación que cualquier desarrollador con conocimientos de UML pueda entender. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, pues ha sido diseñado para modelar los más variados tipos de negocio (Cornejo, 2005).

Se estará haciendo uso del lenguaje UML en su versión 2.1.

1.5 Herramientas para el modelado

Las herramientas CASE (ingeniería de software asistida por computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores (Quintero, y otros, 2012).

1.5.1 Visual Paradigm for UML

Visual Paradigm for UML es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, implementación y pruebas. Ayuda a una rápida construcción de aplicaciones de calidad a un menor costo. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y documentación. Presenta una gama de funcionalidades que lo hacen adaptable y usable para cualquier metodología de desarrollo de software que se utilice. Entre sus características se destacan las siguientes:

- Soporta aplicaciones web.
- Genera informes usables en la generación de documentación.
- Facilita la importación y exportación de ficheros.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones (León, 2000).

Se estará haciendo uso de Visual Paradigm en su versión 5.0.

1.6 Lenguajes de desarrollo del software

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los hombres puedan dar instrucciones a un equipo de cómputo. Consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen la estructura, el significado de sus elementos y expresiones. Un lenguaje de programación permite

a los programadores especificar de manera precisa sobre qué datos una computadora debe operar, cómo deben ser almacenados, transmitidos y qué acciones realizar en presencia de diversas circunstancias (Aranda, 2001).

1.6.1 Lenguajes del lado del cliente

Los lenguajes del lado del cliente son totalmente independientes del servidor. Permiten que la página web pueda ser albergada en cualquier sitio (Instituto Tecnológico de Veracruz, 2007).

Lenguaje de Marcado de Hipertexto

Lenguaje de Marcado de Hipertexto (HTML, de sus siglas en inglés HyperText Markup Language), hace referencia al lenguaje de marcado para la elaboración de páginas web. Define comandos, marcas o etiquetas que permiten delimitar la estructura lógica de un documento web. Para su selección se analizaron características como el hecho de ser un lenguaje sencillo que no necesita de grandes conocimientos cuando se cuenta con un editor web. El texto es estructurado de forma lógica, debido al glosario de etiquetas que provee para la presentación del contenido (Mora, 2002).

Se usará el lenguaje HTML en su versión v5.0.

Hojas de Estilos en Cascada

Hojas de Estilo en Cascada (CSS, de sus siglas en inglés Cascading Style Sheets) hace referencia al lenguaje que describe la presentación de los documentos estructurados en hojas de estilo para diferentes métodos de interpretación, es decir, describe cómo se va a mostrar un documento en pantalla, por impresora, por voz (cuando la información es pronunciada a través de un dispositivo de lectura) o en dispositivos táctiles basados en Braille. El lenguaje CSS se basa en una serie de reglas que rigen el estilo de los elementos en los documentos estructurados y que forman la sintaxis de las hojas de estilo (Alvarez, y otros, 2013).

Se utilizará CSS en su versión v3.0.

JavaScript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Es un lenguaje interpretado por lo que no es necesario compilar los programas para poder ejecutarlos, es decir, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. Se define como basado en prototipos ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM) (Benigni, y otros, 2009).

Se utilizará JavaScript en su versión v1.5.

Lenguaje de Marcado Extensible

La sigla XML es la abreviación de la expresión "Extensible Markup Language", lo que podría traducirse como lenguaje de marcas extensible, que permite definir la gramática de lenguajes específicos para estructurar documentos grandes. Cada paquete de información está delimitado por dos etiquetas como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación. A diferencia de otros lenguajes da soporte a bases de datos, siendo útil cuando varias aplicaciones se deben comunicar entre sí o integrar información. No es un lenguaje en particular sino una manera de definir lenguajes para diferentes necesidades (Paniagua, 1999).

Este lenguaje será utilizado en su versión v1.1.

Tecnologías del lado del cliente

Ajax

Ajax es el acrónimo para Asynchronous JavaScript + XML, que en realidad no es una tecnología sino la combinación de muchas tecnologías como son: HTML, XHTML y CSS para la presentación de la información, el DOM (Document Object Model) y JavaScript que permiten el intercambio directo con la información y XMLHttpRequest que es el protocolo sobre el que está apoyado y que constituye el corazón de Ajax. Utiliza JavaScript para manejar el objeto XMLHttpRequest, el HTML que distribuye en la ventana del navegador los elementos de la aplicación y la información recibida por el servidor, CSS que define el aspecto de cada elemento y dato de la aplicación y XML que no es más que el formato de los datos transmitidos del servidor al cliente y que posteriormente serán mostrados (Fuentes, 200).

En el presente trabajo se utilizará Ajax para la comunicación asincrónica de los datos.

1.6.2 Lenguajes del lado del servidor

Se clasifica así al lenguaje de programación en la arquitectura cliente servidor que se ejecuta del lado del servidor y del cual los usuarios solo obtienen el beneficio del procesamiento de la información (Instituto Tecnológico de Veracruz, 2007).

Procesador de Hipertexto

PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es conocido como una tecnología de código abierto que resulta muy útil para diseñar de forma rápida y eficaz aplicaciones web dirigidas a bases de datos. Su interpretación y ejecución se realizan en el servidor en el cual se encuentra almacenada la página y el cliente solo recibe el resultado de la ejecución. Entre sus ventajas más significativas se pueden presentar las siguientes:

- Multiplataforma.

- Completamente orientado a la web.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad.
- Permite las técnicas de programación orientada a objetos.
- Manejo de excepciones (Álvarez, 2004).

Se utilizará PHP en su versión v5.4.

1.7 Framework

Un *framework* es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual un proyecto de software puede ser más fácilmente organizado y desarrollado (Markiewicz, y otros, 2001).

También es preciso plantear que un *framework* para aplicaciones web es diseñado para apoyar el desarrollo de sitios web dinámicos, aplicaciones y servicios web. Este tipo de *framework* intenta aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web. Por ejemplo, muchos *frameworks* proporcionan bibliotecas para acceder a bases de datos, estructuras para plantillas y gestión de sesiones y con frecuencia facilitan la reutilización de código (Gutierrez, 2010).

Vistos los elementos anteriores se puede destacar el papel protagónico de los *frameworks* en el desarrollo de software.

1.7.1 Frameworks para la capa de presentación

jQuery

jQuery es una biblioteca de *javascript* rápida y concisa que simplifica el recorrido de un documento HTML, de manejo de eventos, animaciones e interacciones Ajax para el desarrollo web rápido (jQuery Community, 2013). Estas características posibilitan que los contenidos se puedan mostrar y se interactúe con los mismos de forma dinámica, brindando la posibilidad de usar este *framework* para apoyar el cumplimiento de las funcionalidades necesarias (Alvarez, 2011).

Se utilizará el *framework* jQuery en su versión v1.10.2.

Bootstrap

El *framework* Bootstrap surgió con el fin de conseguir interfaces de aplicaciones web más amigables con el usuario. Define una serie de etiquetas que posibilitan un maquetado de las interfaces a una mayor precisión. Entre las características que apoyaron su selección se encuentran que: se presenta extensible para disímiles navegadores web, todos los estilos se encuentran en un único archivo y ha sido optimizado para lograr un buen rendimiento de las aplicaciones web, las interfaces de usuario incluyen listado de grupos y paneles, posee un

conjunto de componentes que lo hacen personalizable a cualquier ambiente de desarrollo de software y presenta facilidad de uso por parte de los diseñadores permitiendo un perfecto maquetado del contenido (Cochran, 2012).

Se propone utilizar el *framework* Bootstrap en su versión v3.1.1, debido a que los cambios realizados de una versión a otra no son relevantes.

1.7.2 Frameworks para la capa de lógica de negocio

Symfony

Symfony es un *framework* diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador separando la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web. Automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. Está desarrollado completamente en PHP, ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Proporciona una estructura al código fuente forzando al desarrollador a crear código más legible para un futuro mantenimiento, encapsula operaciones complejas en instrucciones sencillas, es multiplataforma, usa programación orientada a objetos (POO) y soporta integración con el ORM Doctrine y con los *frameworks* de la capa de presentación jQuery y Bootstrap (Potencier, y otros, 2008).

Por todas las características mencionadas anteriormente se selecciona Symfony en su versión v2.3.7 para la implementación de la solución.

1.7.3 Frameworks para la capa de acceso a datos

Mapeador de Objetos Relacional

Un Mapeador de Objetos Relacionales (ORM, de sus siglas en inglés Object Relational Mapper) es una técnica de programación que permite convertir datos entre el sistema de datos utilizado en un lenguaje de programación orientado a objetos y el de una base de datos relacional, es decir, las tablas de la base de datos pasan a ser clases y los registros se pueden manejar con facilidad.

Dentro de las ventajas que condujo a seleccionarlo para desarrollar la solución se encuentran: rapidez en el desarrollo de las aplicaciones, creación automática de un modelo adecuado a partir del esquema de tablas y relaciones, excelente diseño de los códigos generados, ya que se basan en el uso de patrones y buenas prácticas, y se caracterizan por abstraer el motor de

base de datos, lo que significa que se pueda cambiar fácilmente de un sistema gestor a otro sin afectar al resto de la aplicación (Enriquez, y otros, 2011).

Doctrine

Doctrine es un ORM para PHP que proporciona persistencia transparente de los objetos desde el lenguaje PHP. Presenta una serie de características que lo hacen utilizable en numerosos proyectos de desarrollo de software y fomenta su elección para el presente trabajo: soporte para las operaciones habituales de crear, obtener, actualizar y borrar, permite crear manual y automáticamente el modelo de base de datos a implementar y presenta soporte para varios motores de bases de datos (Doctrine Project Team, 2011). El empleo de Doctrine posibilita un trabajo más rápido y eficaz. Posee una amplia documentación lo que facilita la retroalimentación continua y presenta soporte para migraciones.

Se decidió utilizar la versión v2.3 de dicho ORM.

1.8 Sistema gestor de base de datos

Un sistema gestor de base de datos (en inglés DBMS: DataBase Management System) es un sistema de software que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda, ya sea de forma interactiva o a través de un lenguaje de programación. Brinda facilidades eficientes y un grupo de funciones con el objetivo de garantizar la confidencialidad, calidad, seguridad e integridad de los datos que contienen, así como un acceso fácil y eficiente a los mismos (Bertino, y otros, 1995). Entre los DBMS más utilizados se encuentra Oracle, MySQL y PostgreSQL. Para el desarrollo de la aplicación se hará uso de este último pues Oracle no presenta soporte para software libre y MySQL cuenta con dos licencias, una privativa que cuenta con la mayoría del soporte y los servicios mientras que la libre carece de muchos de ellos

1.8.1 PostgreSQL

PostgreSQL es un sistema gestor de base de datos relacional libre y orientado a objetos, publicado bajo la licencia BSD. Se destaca por ejecutar consultas complejas, así como subconsultas de gran tamaño. Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) permite que mientras un proceso escribe en una tabla otros accedan a la misma sin necesidad de bloqueos. Posee una amplia variedad de tipos nativos (números de precisión arbitraria, texto de largo ilimitado, figuras geométricas). Tiene gran soporte para vistas, procedimientos ubicados en el servidor, transacciones, almacenamiento de objetos de tamaño considerable y características orientadas a objetos (Pecos, 2008).

Para el desarrollo de la solución propuesta, se utilizará PostgreSQL en su versión v9.2.4.

1.9 Servidor web

Un servidor web es un programa que permite crear un servidor http en un ordenador. Con el modelo cliente/servidor y el protocolo de transferencia de hipertexto de Internet proporciona a los usuarios archivos que conforman las páginas web (textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de música). Está diseñado para ser un servidor web potente y flexible que pueda funcionar en la más amplia variedad de plataformas y entornos (Sánchez, 2011).

1.9.1 Apache

El servidor Apache es un servidor web HTTP de código abierto. Puede ser usado en varios sistemas operativos, lo que lo hace prácticamente universal. Presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido. Es usado para tareas donde el contenido necesita ser puesto a disposición en una forma segura y confiable. Es posible elegir qué características van a ser incluidas en el servidor seleccionando y qué módulos se van a cargar, ya sea al compilar o al ejecutar el servidor (The Apache Software Foundation, 2012).

Se estará haciendo uso de la versión 2.2.6 del Servidor Web Apache.

1.10 Entorno de desarrollo

1.10.1 PhpStorm

PhpStorm es un entorno de desarrollo integrado (IDE) enfocado en la productividad del desarrollador. Provee completamiento de código, navegación rápida, chequeo de errores al momento, completamiento de código php, detector de código duplicado y mezcla lenguajes (JavaScript, SQL, XML). Está basado en DOM y soporta HTML5. Permite la navegación de código, búsqueda de usos y *debugger* de JavaScript. Es ligero, de fácil instalación, multiplataforma y de código abierto (Taft, 2013).

1.10.2 NetBeans

Este entorno de desarrollo integrado (IDE, de sus siglas en inglés Integrated Development Environment), permite a los desarrolladores escribir, compilar, depurar y ejecutar programas informáticos. Es de código abierto escrito completamente en Java que permite crear aplicaciones de escritorio, web y aplicaciones para móviles utilizando los lenguajes Java, JavaFX, PHP, Java Script, Ruby y Ruby onRails, Groovy y Grails, y C/C++, además de presentar soporte para la tecnología Ajax. Está disponible para múltiples plataformas como son Windows, Mac, Linux y Solaris (Dorado, 2005).

Será utilizado el IDE NetBeans en su versión v8.0 por las características antes mencionadas, destacando además que presenta un entorno de trabajo muy eficiente y hace posible trabajar

con mayor fluidez y facilidad a la hora de implementar productos informáticos. Por otra parte permite la integración con el *framework* Symfony seleccionado para este trabajo, haciendo posible que algunas funcionalidades gestionadas a través de la consola de este marco de trabajo se puedan ejecutar en el propio IDE.

Conclusiones parciales del capítulo

Luego de haber realizado el estudio para dar solución al problema planteado en la presente investigación se obtuvo como resultado la necesidad de elaboración de una herramienta que permita gestionar el proceso de revisiones técnicas formales en el área de Arquitectura de desarrollo del software en el centro FORTES.

El estudio y análisis de las metodologías de desarrollo de software permitió seleccionar XP como la más adecuada para guiar el proceso de desarrollo de la propuesta de solución.

Para el modelado se hará uso del lenguaje de modelado UML y la herramienta Case Visual Paradigm for UML.

Una vez analizadas las herramientas, tecnologías y marcos de trabajo más usados para el desarrollo del software y teniendo en cuenta que en un futuro se pronostica la integración con otras aplicaciones, se determinó como *framework* del negocio Symfony el cual incluye el uso de las tecnologías y herramientas que se enuncian a continuación: NetBeans, Ajax, PostgreSQL, Apache, jQuery, Bootstrap y Doctrine. Además, de los lenguajes seleccionados para la implementación los cuales fueron: HTML, CSS, JavaScript, XML y PHP.

Capítulo 2: Diseño de la propuesta de solución

Introducción

En este capítulo se describe la propuesta de solución del sistema. Se detallan las Historias de Usuario (HU), donde se ha estimado un tiempo para su realización y se ha establecido el orden en que serán implementadas. Como parte del diseño de la propuesta de solución se muestran las tarjetas Clase o Cargo, Responsabilidad, Colaboración (CRC) y los prototipos de interfaz de usuario más importantes del sistema.

2.1 Flujo actual del procedimiento de las RTF en el área de Arquitectura

Dentro del centro FORTES el procedimiento para las RTF en el área de Arquitectura se realiza mediante un flujo de actividades condicionado por diferentes actores u objetos que influyen dentro del mismo. Inicialmente el Proyecto productivo planifica la RTF en su cronograma con una duración de 5 días, posteriormente el Asesor de Calidad del centro solicita la RTF a la Dirección de Calidad, la Dirección de Calidad asigna el responsable para la ejecución de esa RTF. El responsable generalmente es un experto en el tema y se encarga de realizar la RTF, solicitar la documentación de los artefactos de la Arquitectura, revisar la documentación, entrevistar al grupo de arquitectos y emitir el dictamen. A continuación se ejemplifica el flujo de las actividades.

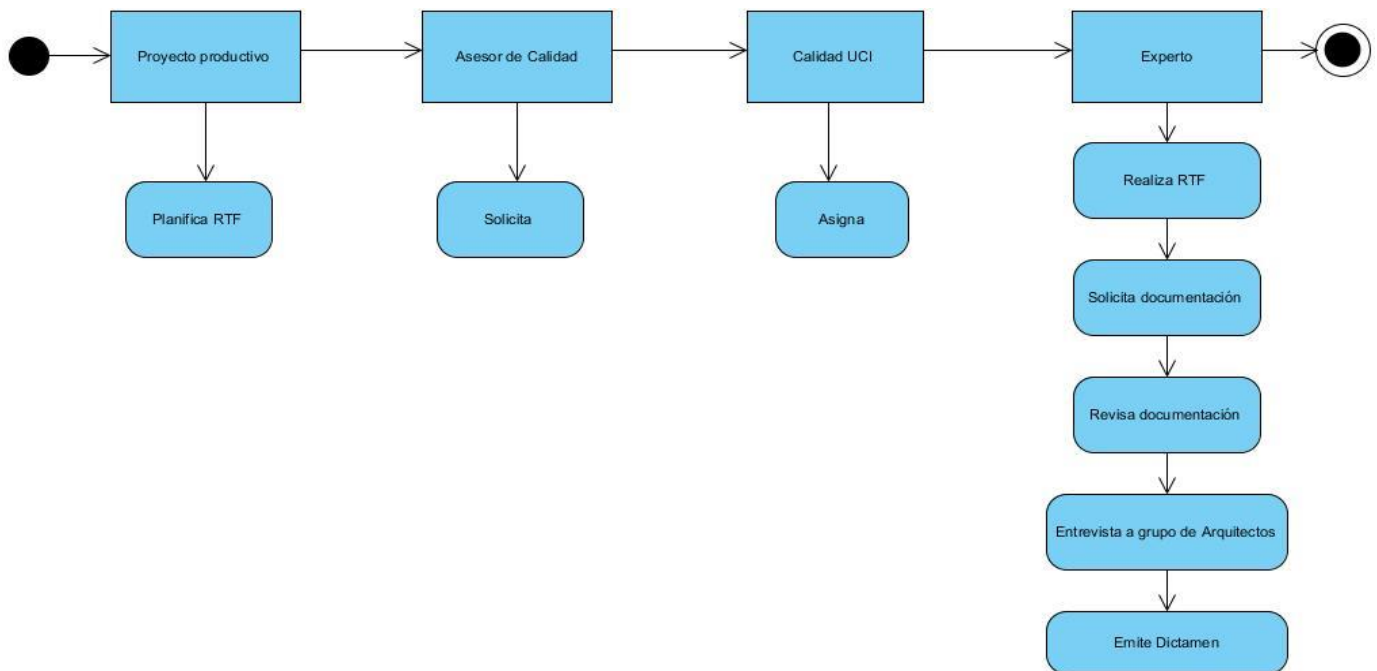


Figura 2: Flujo actual del procedimiento de las RTF en el área de Arquitectura.

2.2 Propuesta de solución

Como solución a la problemática descrita en el presente trabajo se propone la implementación de un sistema que permitirá informatizar el proceso de revisiones técnicas formales en el área arquitectura del desarrollo de software para el centro FORTES.

El procedimiento se informatiza a partir de que comienza la RTF por lo que el experto en el rol de administrador es el que tiene mayor grado de responsabilidad en la interacción con el mismo.

El sistema permitirá fácil acceso a los usuarios sin necesidad de emplear espacio en disco duro para su instalación y uso ya que es un sistema web. Inicialmente contará con dos usuarios, uno con el rol de administrador y otro con rol de usuario para navegación en todas las acciones de su menú lateral, pero sin acceso a la administración de usuarios, expedientes y proyectos, los cuáles serán gestionados solo por el administrador. Proporcionará interfaces sencillas y fáciles de usar que le permitan interactuar de forma cómoda y facilita las acciones a ejecutar. En su interfaz de presentación permitirá la autenticación de los usuarios del sistema a través de una clave de acceso. Brindará la facilidad de registrar los usuarios que harán uso del sistema con sus claves, dirección de correo y el rol a desempeñar durante el proceso de revisión.

En una de sus interfaces principales permitirá al administrador desarrollar el registro de los proyectos a auditar teniendo en cuenta aspectos como nombre, versión y descripción. Dispondrá de una interfaz para el trabajo con las revisiones donde se registrará de cada una de ellas su número, versión, fecha de inicio, fecha de fin y expediente al que se le aplica la revisión. Después de ser creada se habilita un enlace al dictamen técnico. Permitirá gestionar los criterios resultantes de las revisiones tras la evaluación de cada artefacto, documento o vista arquitectónica, así como su seguimiento. Creará un espacio donde se registre el expediente arquitectónico de los proyectos mediante la inserción de la versión, datos del mismo y proyecto que pertenecerá a dicho expediente. Contará con un espacio para el registro de incidencias o no conformidades durante todo el proceso de evaluación de los artefactos, a partir de los errores o deficiencias encontradas en el expediente real de proyecto en el Excriba como Gestor de Documentos Administrativos o repositorio de expedientes de proyecto .Por otra parte se tendrá un espacio para la visualización de reportes generales de la aplicación enmarcados en la cantidad de aspectos de evaluación por vista arquitectónica u artefacto y cantidad de proyectos por tipo de proyecto .

2.3 Exploración

En esta primera fase de la metodología XP se comienza a interactuar con el cliente y el equipo de desarrollo para identificar los requisitos del sistema, así como definir las historias de usuario

(HU). Se determina el número y tamaño de las iteraciones así como, al mismo tiempo el equipo de desarrollo se familiariza con las herramientas y tecnologías que utilizará en el proyecto (Carmona, y otros, 2007).

2.3.1 Historias de usuarios

Las HU son herramientas que dan a conocer los requisitos del sistema al equipo de desarrollo. En ellas el cliente describe la actividad que se realizará en el sistema, muestran la silueta de una tarea a realizar, deben ser claras, sencillas y sin profundizar en detalles. También, son utilizadas para estimar el tiempo que el equipo de desarrollo tomará para realizar las entregas, para estas estimaciones se le asignan unidades de medida a las HU que representen magnitud y no días reales de duración, tales como: puntos de estimación, días ideales u otra unidad de medida (Carmona, y otros, 2007).

Seguidamente se detallan los campos que componen las HU en función:

- Número de HU: este campo contiene el número que le es asignado a las HU consecutivamente.
- Nombre de HU: identifica la HU a la que se hace alusión.
- Usuario: usuario del sistema que interactúa o realiza la funcionalidad.
- Prioridad en el negocio: prioridad que le es asignada a la HU en el negocio de acuerdo a las necesidades del usuario. Los valores que puede tomar son Alta, Media o Baja.
- Riesgo en el desarrollo: riesgo que se corre en el desarrollo de la aplicación en caso de no realizarse la HU. Los valores que puede tomar son Alto, Medio o Bajo.
- Puntos estimados: esta estimación se realiza en semanas trabajando 8 horas diarias de lunes a viernes, la misma es formalizada por el equipo de desarrollo evaluando el tiempo que incurre en la realización de esta HU.
- Iteración asignada: este campo es en dependencia de la prioridad que se establece, lo cual se determina en la iteración que se desarrolla la HU.
- Descripción: como su nombre lo indica, se realiza una pequeña descripción de la función de la HU.
- Observación: datos adicionales de la HU en cuestión, resalta datos que son necesarios cumplir, para una mejor utilización y entendimiento de la funcionalidad que se brinda.

Como resultado del trabajo realizado durante la fase de exploración se identificaron un total de veintidós historias de usuario, a continuación se muestran algunas de ellas (Ver más anexo 1):

Tabla 1: HU Revisar documentación.

Historia de usuario

Nombre: Revisar documentación	Número: 18
Usuarios: usuario	Iteración asignada: 3
Prioridad en el negocio: alta	Puntos estimados: 1
Riesgo en el desarrollo: alto	
Descripción: permite revisar la documentación del proyecto de acuerdo a los aspectos de revisión de la arquitectura.	
Observaciones: el usuario debe estar autenticado en el sistema y se debe tener almacenado el Expediente de Arquitectura del proyecto.	

Tabla 2: HU Crear evaluación.

Historia de usuario	
Nombre: Crear evaluación	Número: 19
Usuarios: usuario	Iteración asignada: 3
Prioridad en el negocio: alta	Puntos estimados: 0.5
Riesgo en el desarrollo: bajo	
Descripción: permite crear una evaluación con los resultados obtenidos en la RTF.	
Observaciones: el usuario debe estar autenticado en el sistema y debe haberse creado al menos una RTF con anterioridad.	

Tabla 3: HU Almacenar criterios.

Historia de usuario	
Nombre: Almacenar criterios	Número: 20
Usuarios: usuario	Iteración asignada: 3
Prioridad en el negocio: media	Puntos estimados: 0.5

Riesgo en el desarrollo: bajo
Descripción: se pueden almacenar los criterios de los expertos en RTF en al área de Arquitectura.
Observaciones: el usuario debe estar autenticado en el sistema y debe haberse creado al menos una RTF con anterioridad.

2.3.2 Características del sistema

Los requerimientos no funcionales son restricciones de los servicios o funciones ofrecidos por un sistema. No se refieren directamente a las funcionalidades específicas que este proporciona, sino a sus propiedades emergentes como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (Sommerville, 2011).

Las características con las que debe contar la solución fueron definidas por el equipo de desarrollo y el cliente como se manifiesta a continuación:

Usabilidad

El sistema podrá ser usado por personas con conocimientos básicos en el manejo de computadoras. Se emplearán barras de progreso para indicar el estado de los procesos que por su complejidad requieran de un tiempo de procesamiento apreciable por los usuarios. El software tendrá siempre visible la opción de Ayuda, lo que posibilitará un mejor aprovechamiento por parte de los usuarios de sus funcionalidades.

Rendimiento

Teniendo en cuenta que el producto se debe diseñar sobre una arquitectura cliente - servidor, los tiempos de respuestas del sistema deben ser rápidos, al igual que la velocidad de procesamiento de la información.

Restricciones de diseño

El producto de software final debe diseñarse sobre una arquitectura cliente-servidor. Se deben emplear los estándares existentes para el diseño de interfaces, base de datos y codificación.

Requerimiento de ayuda y documentación

El sistema debe contar con una ayuda general en la página principal, para guiar al usuario a trabajar con el sistema.

Interfaz

El sistema debe contar con una interfaz fácil de usar, sencilla, permitiendo que los usuarios sean capaces de interactuar con la aplicación aun teniendo conocimientos básicos de informática. Será diseñada utilizando colores refrescantes, agradables y se emplearán imágenes identificadas con el negocio del sistema.

Portabilidad

El sistema será un sistema multiplataforma y debe ser compatible con los sistemas operativos Windows y Linux.

Seguridad

El usuario debe autenticarse antes de entrar al sistema.

Confiabledad: la información que se maneje en el sistema estará protegida de acceso no autorizado y divulgación, a partir de los diferentes roles de los usuarios que empleen el sistema.

Integridad: la información manejada por el sistema será objeto de cuidadosa protección contra corrupción y estados inconsistentes, de igual manera el origen y autoridad de los datos.

Disponibilidad: la información se encontrará disponible en todo momento para aquellos usuarios autorizados a acceder al sistema.

2.4 Planificación

En esta fase los clientes establecen la prioridad de las HU en correspondencia con las necesidades inmediatas para poder organizarlas en iteraciones. Se realiza una estimación del esfuerzo que se requiere por parte del equipo para implementarlas; en la que se utiliza como medida el punto, un punto equivale a una semana ideal de programación. Esta fase crea visibilidad de tal manera que se puede ver la duración del proyecto (Procesos de Software, 2014).

2.4.1 Estimación de esfuerzo por historia de usuario

Se realizó una estimación de esfuerzo para cada una de las historias de usuario, en la siguiente tabla se muestra el número, nombre y esfuerzo asignado a cada una.

Tabla 4: Estimación del esfuerzo.

Número	Nombre	Estimación(semanas)
1	Registrar usuarios	0.5
2	Mostrar datos del usuario	0.1

3	Editar usuario	0.3
4	Eliminar usuario	0.2
5	Autenticar usuario	0.5
6	Registrar proyecto	0.5
7	Mostrar proyecto	0.1
8	Editar proyecto	0.3
9	Eliminar proyecto	0.2
10	Adicionar incidencia	0.5
11	Mostrar incidencia	0.1
12	Editar incidencia	0.3
13	Eliminar incidencia	0.2
14	Determinar complejidad de la incidencia	0.3
15	Determinar impacto de la incidencia	0.5
16	Determinar estado de la solución	0.4
17	Solicitar documentación	1
18	Revisar la documentación	1
19	Crear evaluación	0.5
20	Almacenar criterios de expertos	0.5
21	Revisar las evidencias	0.2
22	Emitir el dictamen	0.4

La estimación del esfuerzo total es de 8.6 semanas.

2.4.2 Plan de iteraciones

Las iteraciones son un conjunto de HU que se van a implementar y como resultado arrojan la entrega de un módulo del proyecto. El plan de iteraciones establece cuántas iteraciones serán necesarias realizar sobre el sistema para su consecución y las HU que se desarrollarán en cada una. Tomando como referencia los aspectos antes tratados y la aplicación que se pretende construir se realizarán 3 iteraciones.

Primera iteración: permite que el usuario pueda registrarse, autenticarse y gestionar sus datos. Se da cumplimiento al desarrollo de las HU número 1, 2, 3, 4 y 5

Segunda iteración: consiste en la gestión de proyecto a los que se les aplicará una RTF y de las incidencias detectadas. Se da cumplimiento al desarrollo de las HU número 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 y 16.

Tercera iteración: radica en la gestión de la documentación asociada al expediente de Arquitectura de un proyecto. Se da cumplimiento al desarrollo de las HU número 17,18, 19, 20, 21 y 22.

Tabla 5: Plan de iteraciones.

Iteraciones	Orden de las HU	Duración total
Iteración 1	Registrar usuarios	1.6 semanas
	Mostrar datos del usuario	
	Editar usuario	
	Eliminar usuario	
	Autenticar usuario	
Iteración 2	Registrar proyecto	3.4 semanas
	Mostrar proyecto	
	Editar proyecto	
	Eliminar proyecto	
	Adicionar incidencia	
	Mostrar incidencia	

	Editar incidencia	
	Eliminar incidencia	
	Determinar complejidad de la incidencia	
	Determinar impacto de la incidencia	
	Determinar estado de la solución de la incidencia	
Iteración 3	Solicitar documentación	3.6 semanas
	Revisar la documentación	
	Crear resumen	
	Almacenar criterios de expertos	
	Revisar las evidencias	
	Emitir el dictamen	

2.5 Diseño

El diseño en XP establece prácticas especializadas que inciden directamente en la realización del diseño para lograr un sistema robusto y reutilizable tratando de mantener su simplicidad, es decir, crear un diseño evolutivo que se va mejorando incrementalmente y que permite hacer entregas pequeñas y frecuentes de valor para el cliente. A la hora de darle cumplimiento a la actividad de diseñar, XP no especifica ninguna técnica de modelado, se utilizan sencillos esquemas en una pizarra, denominados diagramas de clases utilizando UML o tarjetas CRC (Clase, Responsabilidad y Colaboración) siempre que sean útiles, tributen a la comprensión y no requieran mucho tiempo en su creación (Metodología XP, 2014).

2.5.1 Tarjetas CRC

Las tarjetas CRC se elaboran con el objetivo de realizar un mejor diseño de la propuesta de solución. Estas tarjetas se dividen en tres secciones que contienen la información del nombre de la clase, sus responsabilidades y las clases que le brindan soporte. Su principal utilidad es

dejar el enfoque procedimental y entrar al modelo orientado a objetos (Carmona, y otros, 2007). Seguidamente se detallan los campos que componen las tarjetas CRC en función:

- Clase: nombre de la clase con que se está modelando.
- Responsabilidades: descripción breve del propósito de la clase.
- Colaboraciones: indica las relaciones entre clases para cumplir con la responsabilidad.

Como resultado del trabajo realizado durante la fase de diseño se obtuvieron un total de cuatro tarjetas CRC las cuales se muestran a continuación.

Tabla 6: Tarjeta CRC tbl_proyecto.

Clases: tbl_proyecto	
Responsabilidades:	Colaboraciones:
Registrar proyecto	tbl_tipo_proyecto
Mostrar proyecto	
Editar proyecto	
Eliminar proyecto	

Tabla 7: Tarjeta CRC tbl_no_conformidad.

Clases: tbl_no_conformidad	
Responsabilidades:	Colaboraciones:
Registrar incidencia	tbl_informe_nc
Mostrar incidencia	tbl_complejidad_nc
Editar incidencia	tbl_dictamen_rtf
Eliminar incidencia	tbl_estado_solucion

Determinar complejidad de la incidencia	tbl_impacto_nc
Determinar impacto de la incidencia	
Determinar estado de solución de la incidencia	

Tabla 8: Tarjeta CRC tbl_evaluacion.

Clases: tbl_evaluacion	
Responsabilidades:	Colaboraciones:
Solicitar documentación	tbl_documento_arquitectura
Revisar la documentación	tbl_aspectos
Crear evaluación	
Almacenar criterios	

Tabla 9: Tarjeta CRC tbl_rtf_arquitectura.

Clases: tbl_rtf_arquitectura	
Responsabilidades:	Colaboraciones:
Revisar evidencias	tbl_expediente_arquitectura

Emitir dictamen

tbl_aspectos

2.5.2 Modelo de datos

Un modelo de datos es la representación abstracta de los datos en un sistema gestor de base de datos, está formado por tres elementos fundamentales:

- Objetos (entidades que existen y se manipulan).
- Atributos (características básicas de estos objetos).
- Relaciones (forma en que se enlazan los distintos objetos entre sí).

Para su construcción se tuvo en cuenta todos los datos que deben persistir en el sistema y en los nomencladores se agruparon los estándares predeterminados del negocio del proceso. A continuación se muestra el modelo de datos que se obtuvo a partir de las clases generadas en las tarjetas CRC:

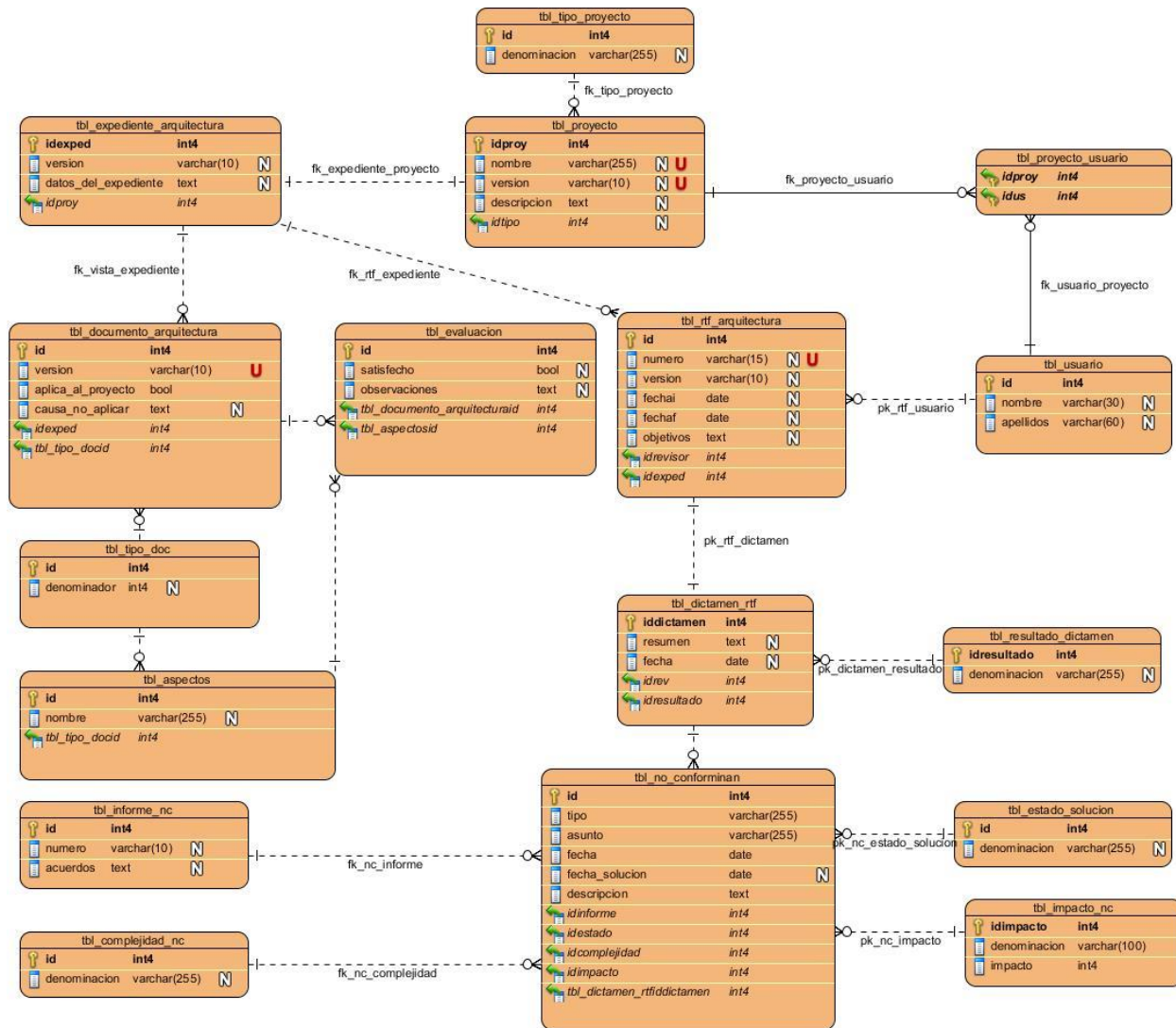


Figura 3: Modelo de datos.

2.5.3 Prototipos de interfaz de usuario

Los prototipos funcionales permiten al usuario tener una idea de las interfaces que mostrará el sistema, para alcanzar una retroalimentación sobre los requerimientos del mismo. Son el esqueleto de la interfaz de usuario y tienen el propósito de probar el diseño de las interfaces, incluyendo la usabilidad que estas pueden tener antes de que se comience con el desarrollo del software. De esta manera se valida que se esté cumpliendo con los requerimientos exigidos antes de que se realice todo el esfuerzo necesario para el desarrollo (Centro Nacional de Tecnologías de Información, 2012). A continuación se muestran algunos de los prototipos funcionales de interfaz de usuario diseñados:

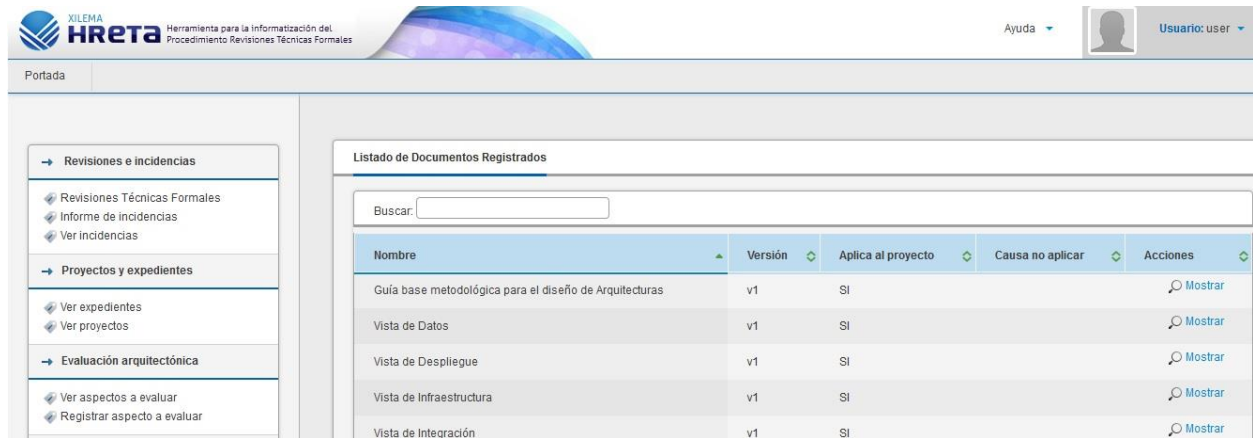


Figura 4: Diseño de interfaz gestionar documentación.

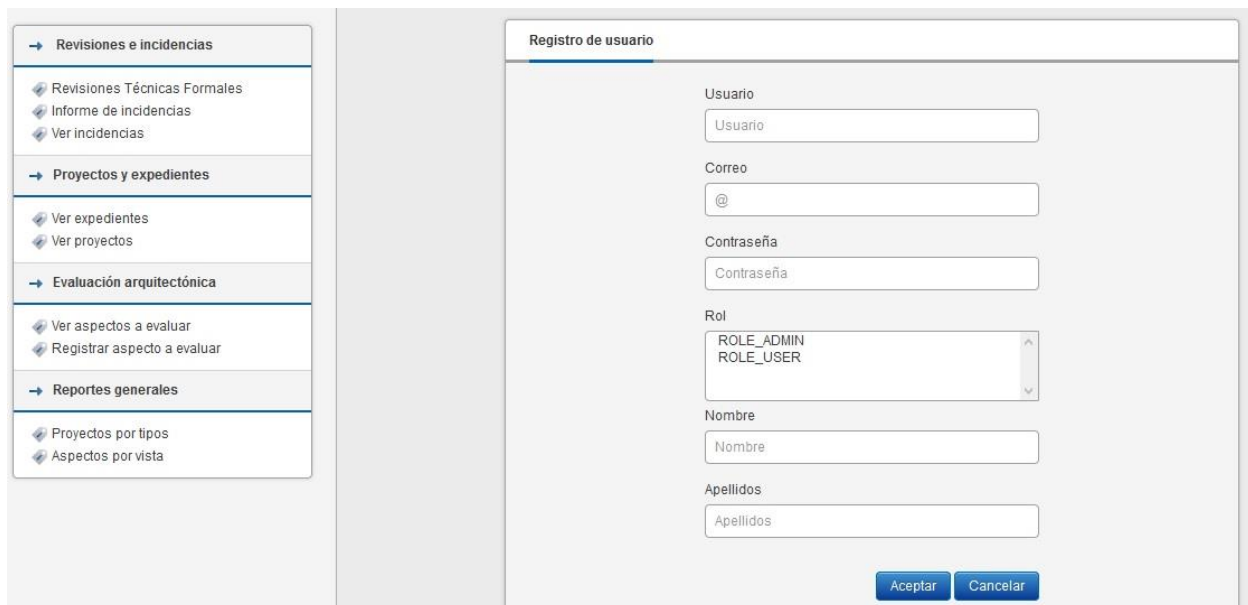


Figura 5: Diseño de interfaz gestionar usuario.



Figura 6: Diseño de interfaz autenticar usuario.

2.5.4 Patrón arquitectónico empleado

El patrón Modelo Vista Controlador (MVC), empleado en la solución, define responsabilidades y dependencias obedeciendo a objetivos específicos representados por tres paradigmas. Separa los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos, permitiendo mayor independencia, mantenimiento y reutilización.

Modelo: representa los datos del programa y controla todas sus transformaciones.

Para la aplicación propuesta una de las tareas más comunes es la persistencia y lectura de la base de datos, para lo que emplea las funcionalidades de Doctrine que puede persistir y recuperar objetos completos de la base de datos. Este funcionamiento se logra asociando la clase Proyecto.php a la tabla tbl_proyecto de la base de datos y las propiedades de dicha clase a las columnas de la tabla; además del uso de un repositorio ProyectoRepository.php asociado a la entidad que contiene las consultas a la base de datos.

```
<?php

namespace HRETA\EntidadesBundle\Repository;

use Doctrine\ORM\EntityRepository;

class ProyectoRepository extends EntityRepository {

    public function getProyectos() {
        $em = $this->getEntityManager();
        $query = $em->createQueryBuilder()
            ->select('p')
            ->from('EntidadesBundle:Proyecto', 'p')
            ->leftJoin('p.idexped', 'ea')
            ->where('ea.id IS NULL')
        ;
        return $query;
    }
}
```

Figura 7: Clase ProyectoRepository.

```
<?php

namespace HRETA\EntidadesBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

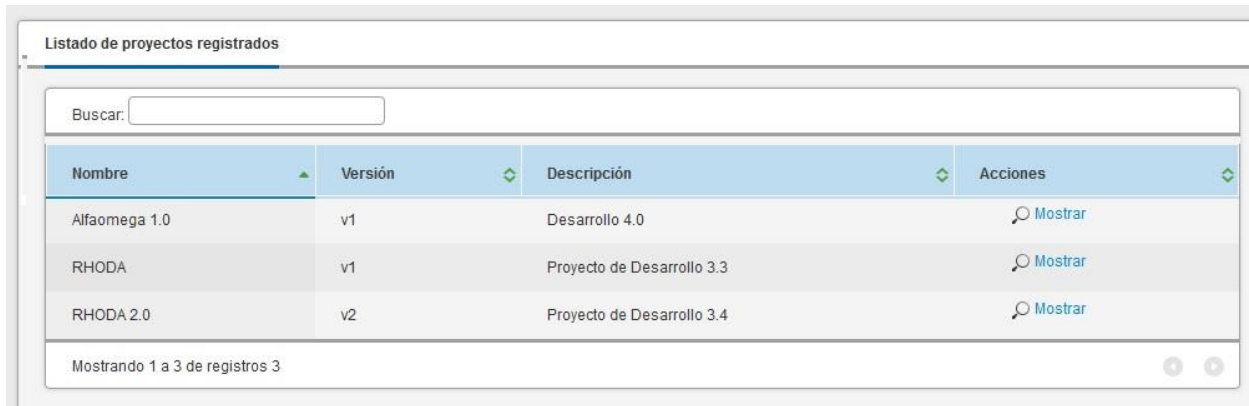
/**
 * Proyecto
 *
 * @ORM\Table(name="tbl_proyecto")
 * @ORM\Entity
 * @ORM\Entity(repositoryClass="HRETA\EntidadesBundle\Repository\ProyectoRepository")
 */
class Proyecto {

    /**
     * @var integer
     *
     * @ORM\Column(name="idproy", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="SEQUENCE")
     * @ORM\SequenceGenerator(sequenceName="tbl_proyecto_idproy_seq", allocationSize=1, initialValue=1)
     */
    private $id;
```

Figura 8: Declaración de la clase Proyecto.

Vista: genera la presentación visual de los datos representados por el Modelo y muestra los datos al usuario.

Twig es un motor y lenguaje de plantillas para PHP muy rápido y eficiente. Symfony 2 recomienda utilizar Twig para crear las plantillas de la aplicación. La sintaxis de Twig se ha diseñado para que las plantillas sean concisas y fáciles de leer y escribir. En la solución se utiliza la herencia a dos niveles para reutilizar el máximo código posible. En el primer nivel se encuentra una sola plantilla base de la aplicación llamada `base.html.twig` y en el segundo nivel plantillas específicas para cada elemento de la aplicación.



Nombre	Versión	Descripción	Acciones
Alfaomega 1.0	v1	Desarrollo 4.0	Mostrar
RHODA	v1	Proyecto de Desarrollo 3.3	Mostrar
RHODA 2.0	v2	Proyecto de Desarrollo 3.4	Mostrar

Mostrando 1 a 3 de registros 3

Figura 9: Interfaz de usuario diseñada con el uso de Twig.

Controlador: maneja las entradas del usuario, actuando sobre los datos representados por el Modelo (Mestras, 2009).

En la aplicación los controladores son funciones php que toman información de la petición HTTP y construyen una respuesta HTTP como un objeto de Symfony 2. Los controladores tienen la lógica que el componente necesita para reproducir el contenido de las páginas. La clase controladora `ProyectoController.php` tiene varias acciones, por ejemplo `indexAction` que se encarga de leer información de la petición y mostrar todos los proyectos almacenados en la base de datos.


```
<?php

namespace HRETA\EntidadesBundle\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use HRETA\EntidadesBundle\Entity\Proyecto;
use HRETA\EntidadesBundle\Form\ProyectoType;
use HRETA\EntidadesBundle\Util\FlashMessageManager;

class ProyectoController extends Controller {

    public function indexAction() {
        $em = $this->getDoctrine()->getManager();
        $request = $this->get('request');
        if ($request->query->has("reject")) {
            $this->get('session')->getFlashBag()->add(
                'notice', FlashMessageManager::RejectedFlash
            );
        }
        $entities = $em->getRepository('EntidadesBundle:Proyecto')->findAll();
        $deleteForm = $this->createDeleteForm(-1);
        return $this->render('EntidadesBundle:Proyecto:index.html.twig', array(
            'entities' => $entities,
            'delete_form' => $deleteForm->createView(),
        ));
    }
}
```

Figura 10: Clase controladora ProyectoController.

2.5.5 Patrones de diseño

Los patrones de diseño son una solución estándar para un problema común de programación, constituyen una descripción de clases y objetos comunicándose entre sí, adaptados para resolver un problema de diseño general en un contexto particular. Promueven la reutilización y agilizan el proceso de desarrollo de software. En principio, se aplican solo en la fase de diseño, aunque se ha comenzado a definir y aplicar patrones en otras etapas del proceso de desarrollo, desde la concepción arquitectónica inicial hasta la implementación del código (Prieto, 2009).

A continuación se explican los patrones de diseño seleccionados para ser usados en el desarrollo de la solución.

Patrones generales de software para asignar responsabilidades (GRASP)

Experto: se encarga de asignar la responsabilidad al experto en la información, la clase que cuenta con la información necesaria para cumplir la responsabilidad. Permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les

pide, lo que provee un bajo nivel de acoplamiento. Promueve clases sencillas y cohesivas que son más fáciles de mantener y comprender.

El patrón experto fue utilizado en la capa de abstracción del modelo. Las clases generadas poseen un grupo de funcionalidades que facilitan el acceso y la manipulación de los datos de las entidades persistentes en la base de datos. A continuación se presenta un ejemplo con la clase entidad Proyecto.php:

```
<?php

namespace HRETA\EntidadesBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Proyecto
 *
 * @ORM\Table(name="tbl_proyecto")
 * @ORM\Entity
 * @ORM\Entity(repositoryClass="HRETA\EntidadesBundle\Repository\ProyectoRepository")
 */
class Proyecto {

    /**
     * @var integer
     *
     * @ORM\Column(name="idproy", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="SEQUENCE")
     * @ORM\SequenceGenerator(sequenceName="tbl_proyecto_idproy_seq", allocationSize=1, initialValue=1)
     */
    private $id;
```

Figura 11: Declaración de la clase Proyecto.

Creador: consiste en asignar a un objeto la responsabilidad de crear otro objeto. La nueva instancia deberá ser creada por la clase que: tiene la información necesaria para realizar la creación del objeto, usa directamente las instancias creadas del objeto, almacena y/o maneja varias instancias de la clase o contiene y/o agrega la clase.

En Symfony 2 en las clases controladoras se definen y ejecutan las acciones. En las acciones se crean los objetos de las clases que representan las entidades evidenciando que las clases controladoras son creadoras de dichas entidades.

```
public function createAction(Request $request) {
    $entity = new Proyecto();
    $form = $this->createCreateForm($entity);
    $form->handleRequest($request);

    if ($form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $em->persist($entity);
        $em->flush();
        $this->get('session')->getFlashBag()->add(
            'notice', FlashMessageManager::CreateFlash
        );

        return $this->redirect($this->generateUrl('proyecto_show', array(
            'id' => $entity->getId())));
    }

    return $this->render('EntidadesBundle:Proyecto:new.html.twig', array(
        'entity' => $entity,
        'form' => $form->createView(),
    ));
}
```

Figura 12: Función donde se crea un objeto de la clase Proyecto.

Controlador: sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control. Ofrece una guía para tomar decisiones sobre los eventos de entrada, asignando la responsabilidad del manejo de mensajes de los eventos del sistema a una clase controladora.

En Symfony 2 el patrón Controlador es utilizado en las clases de la capa del controlador del patrón MVC, como son: ProyectoController.php y UsuarioController.php.

```
<?php

namespace HRETA\EntidadesBundle\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use HRETA\EntidadesBundle\Entity\Proyecto;
use HRETA\EntidadesBundle\Form\ProyectoType;
use HRETA\EntidadesBundle\Util\FlashMessageManager;

class ProyectoController extends Controller {

    public function indexAction() {
        $em = $this->getDoctrine()->getManager();
        $request = $this->get('request');
        if ($request->query->has("reject")) {
            $this->get('session')->getFlashBag()->add(
                'notice', FlashMessageManager::RejectedFlash
            );
        }
        $entities = $em->getRepository('EntidadesBundle:Proyecto')->findAll();
        $deleteForm = $this->createDeleteForm(-1);
        return $this->render('EntidadesBundle:Proyecto:index.html.twig', array(
            'entities' => $entities,
            'delete_form' => $deleteForm->createView(),
        ));
    }
}
```

Figura 13: Clase controladora ProyectoController.

Bajo acoplamiento: el acoplamiento es una medida de la fuerza con que una clase está conectada a otras. Este patrón da soporte a una mínima dependencia entre clases y a un aumento de la reutilización. De este modo una modificación en alguna de las clases tiene la mínima repercusión posible en el resto de las clases.

En la solución la clase ProyectoController.php hereda de Controller.php, que es una clase estable en cuanto a su implementación por lo que se obtiene un grado bajo de acoplamiento entre clases. Además, al no asociar las clases del modelo con las de la vista o el controlador, la dependencia entre las clases se mantiene baja.

Alta cohesión: la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realizan un trabajo enorme. Fomenta la reutilización mejorando la claridad y facilidad del diseño.

En la aplicación se evidencia la alta cohesión en la implementación de las clases entidades (Prieto, 2009).

Patrones Gang of Four (GOF)

Fachada: es un patrón estructural que establece un objeto fachada proporcionando una interfaz única y simplificada para los servicios más generales del subsistema. Permite reducir la complejidad de diseño del sistema, minimizando la comunicación y las dependencias.

En Symfony 2 se puede acceder al manejador de entidades de las formas siguientes: `$this->getDoctrine()->getEntityManager()` y `$this->container->get('doctrine')->getEntityManager()` lo que evidencia el uso de este patrón.

```
public function indexAction() {
    $em = $this->getDoctrine()->getManager();
    $request = $this->get('request');
    if ($request->query->has("reject")) {
        $this->get('session')->getFlashBag()->add(
            'notice', FlashMessageManager::RejectedFlash
        );
    }
}
```

Figura 14: Manejo de entidades.

Patrón decorador: añade responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades.

En el sistema se utiliza el patrón decorador para la vista, el `base.html.twig` decora el contenido de la plantilla.

Conclusiones parciales del capítulo

Luego de realizar el estudio del negocio que abarca la investigación se elaboró la propuesta de solución teniendo en cuenta los aportes de los sistemas estudiados.

Se identificaron veintidós historias de usuario las cuales permitieron conocer los requisitos y características de la propuesta de solución.

Se elaboró un plan de iteraciones y se efectuó la estimación del esfuerzo determinándose la realización de tres iteraciones en un tiempo total de 8.6 semanas.

Se elaboraron cuatro tarjetas CRC que sirvieron como base para el diseño del modelo de datos de la solución.

Se diseñaron los prototipos de interfaz de usuario que constituyen un elemento base para el inicio del proceso de implementación.

Capítulo 3: Implementación y pruebas de la propuesta

Introducción

En el presente capítulo se describe el proceso de implementación y pruebas de la propuesta de solución, guiados por la metodología de desarrollo seleccionada. Se hace referencia al estándar de codificación utilizado para el desarrollo de la propuesta de solución. Se presentan las tareas de ingeniería cuyas descripciones guían el proceso de implementación. Además, se muestran los resultados de las pruebas unitarias y de aceptación realizadas al producto.

3.1 Mapa de navegación

El modelo de navegación está compuesto por uno o varios mapas de navegación que representan y estructuran la visión global del sistema, estos se elaboran según la cantidad que se desee ya que depende de la dificultad del producto, se representan usando un grafo dirigido en el cual los nodos constituyen las pantallas del sistema y los arcos son los enlaces de navegación.

En este nivel de abstracción, solo es de interés especificar qué pantallas conformarán el mapa de navegación y desde dónde serán alcanzables.

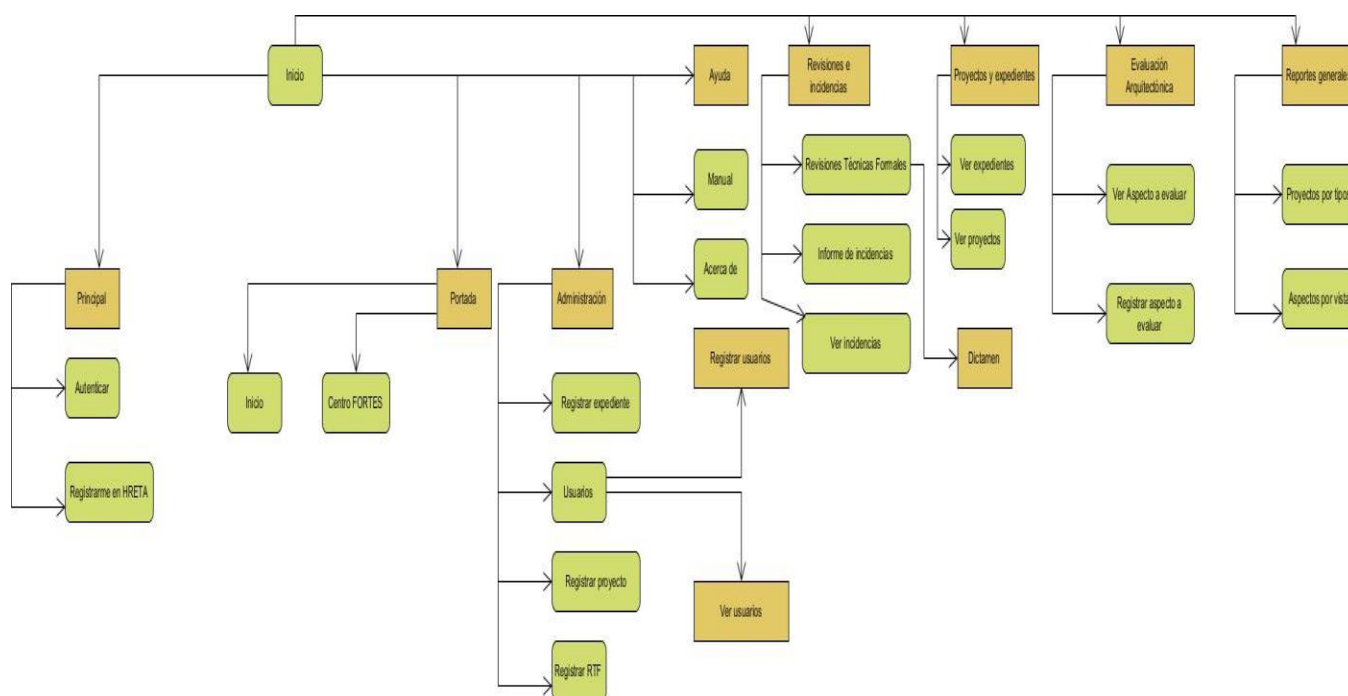


Figura 15: Mapa de navegación.

3.2 Estándar de codificación

Un estándar de codificación comprende los aspectos de la generación de código. Al inicio de un desarrollo de software, es necesario establecer un estándar de codificación para asegurar que

los programadores del proyecto trabajen de forma coordinada. Establecer un estándar de codificación asegura que un equipo de programadores mantenga un código de calidad (Díaz, y otros, 2012). A continuación se enumeran algunos de ellos:

- Se utiliza el tabulador para la alineación, excepto en los archivos YAML donde se utilizan dos espacios.
- Para el caso de las tablas en las base de datos debe agregar el prefijo *tbl_*, por ejemplo si una tabla se llama *comments* en la base de datos se le nombra *tbl_comments*.
- Añadir una coma después de cada elemento del arreglo en un arreglo multilínea, incluso después del último.
- Declara las propiedades de clase antes que los métodos.
- Debe haber una línea en blanco después de los espacios de nombres y debajo de los bloques “use”.
- Los nombres de las propiedades no deben tener como prefijo el caracter “_” para indicar la visibilidad *protected* o *private*.
- Debe haber un espacio después de la palabra clave de la estructura de control.

3.3 Implementación

La implementación se hace atendiendo a los estándares de codificación y de forma paralela con el diseño. En esta fase se propone tener en cuenta aspectos muy importantes como la programación en equipo y la disponibilidad del cliente para lograr mayores resultados en la implementación del software.

Disponibilidad del cliente: formar parte del equipo de desarrollo ayuda a solucionar todas las dudas que puedan surgir y así se asegura que lo implementado cubre con las dudas planteadas.

Desarrollo en pareja: toda la implementación fue realizada por dos personas que trabajaron de forma conjunta para llevar a cabo la realización del producto de software en tiempo y forma. Esto tiene como ventaja que se tenga un diseño de mejor calidad, un código organizado y con menos errores.

3.3.1 Tareas de ingeniería

Para llevar a cabo la correcta implementación de las HU se definen por parte del equipo de desarrollo las tareas de Ingeniería (TI), que se realizarán 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, permitiéndole la representación gráfica de las responsabilidades asignadas a cada miembro del equipo de desarrollo (Ariel Erlijman Piwen, 2001).

A continuación se muestran ejemplos de las tareas de ingeniería (Ver más anexo 2) pertenecientes a la fase de implementación.

Tabla 10: TI Revisar documentación.

Tarea de ingeniería	
No. de la tarea: 18	No. de la HU: 18
Nombre de la tarea: Revisar documentación	
Tipo de tarea: desarrollo	Puntos estimados: 1
Programador responsable: Yairon Consuegra Cabrera	
Descripción: se implementa la funcionalidad revisar documentación. Mediante la implementación de esta tarea se revisa la documentación del proyecto y es donde se detectan las incidencias. El usuario selecciona la opción revisar documentación de acuerdo a los aspectos de revisión de la arquitectura	

Tabla 11: TI Crear evaluación.

Tarea de ingeniería	
No. de la tarea: 19	No. de la HU: 19
Nombre de la tarea: Crear evaluación	
Tipo de tarea: desarrollo	Puntos estimados: 0.5
Programador responsable: Yairon Consuegra Cabrera	
Descripción: se implementa la funcionalidad crear evaluación. Mediante la implementación de esta tarea se crea una evaluación que contiene los resultados obtenidos en la realización de la RTF a un documento de la arquitectura de un proyecto. El usuario selecciona el documento y genera la evaluación del mismo.	

Tabla 12: TI Almacenar criterios.

Tarea de ingeniería	
No. de la tarea: 20	No. de la HU: 20
Nombre de la tarea: Almacenar criterios	
Tipo de tarea: desarrollo	Puntos estimados: 0.5
Programador responsable: Gino Miguel Ricardo González	
Descripción: se implementa la funcionalidad almacenar criterios. Mediante la implementación de esta tarea almacenan los criterios de expertos en RTF en el área de Arquitectura. El usuario selecciona la opción almacenar criterios y es guardado el criterio del experto para su posterior reutilización.	

3.4 Pruebas

Las pruebas constituyen una de las fases de la metodología XP, perfiladas con el objetivo de comprobar el funcionamiento del código que se ha implementado. Constituyen una prueba formal que permite al usuario, cliente u otra entidad autorizada, determinar la aceptación de un sistema o componente (Institute of Electrical and Electronics Engineers, 1990) .

En XP se definen dos tipos de pruebas:

Pruebas de aceptación o funcionales: son pruebas específicas, concretas y exhaustivas para probar y validar que el software hace lo que debe y sobre todo, lo que se ha especificado.

Pruebas unitarias: diseñadas por los programadores para verificar el código (Gutiérrez, 2010).

3.4.1 Pruebas de aceptación

En la presente investigación las pruebas de aceptación fueron creadas teniendo como base las HU. Se definieron las entradas al sistema y los resultados esperados ante estas entradas. A continuación se muestran los casos de pruebas de aceptación de las funcionalidades más críticas del sistema (Ver más anexo 3).

Tabla 13: Caso de prueba de aceptación revisar documentación.

Caso de prueba de aceptación	
Código: 18	HU: 18
Nombre: Revisar documentación	

Descripción: prueba para la funcionalidad que permite revisar la documentación del proyecto de acuerdo a los aspectos de revisión de la arquitectura.
Condiciones de ejecución: el usuario debe estar autenticado en el sistema y se debe tener almacenado el Expediente de Arquitectura del proyecto.
Entradas/Pasos de ejecución: revisar la documentación del proyecto de acuerdo a los aspectos de revisión de la arquitectura.
Resultado esperado: detectar evidencias en el Expediente de Arquitectura de un proyecto.
Evaluación de la prueba: satisfactoria

Tabla 14: Caso de prueba de aceptación crear evaluación.

Caso de prueba de aceptación	
Código: 19	HU: 19
Nombre: Crear evaluación	
Descripción: prueba para la funcionalidad que permite crear una evaluación con los resultados obtenidos en la RTF.	
Condiciones de ejecución: el usuario debe estar autenticado en el sistema y debe haberse realizado al menos una revisión con anterioridad.	
Entradas/Pasos de ejecución: se selecciona el documento y genera la evaluación.	
Resultado esperado: que se cree una evaluación que contenga los resultados obtenidos en la realización de la RTF a un documento de la arquitectura de un proyecto.	
Evaluación de la prueba: satisfactoria	

Tabla 15: Caso de prueba de aceptación almacenar criterios.

Caso de prueba de aceptación

Código: 20	HU: 20
Nombre: Almacenar criterios	
Descripción: prueba para la funcionalidad que permite almacenar los criterios de los expertos en RTF en el área de Arquitectura.	
Condiciones de ejecución: el usuario debe estar autenticado en el sistema y debe haberse realizado al menos una revisión con anterioridad.	
Entradas/Pasos de ejecución: se selecciona la opción almacenar criterios y es guardado el criterio del experto para su posterior reutilización.	
Resultado esperado: que se almacenen en el sistema los criterios de los expertos en las RTF.	
Evaluación de la prueba: satisfactoria	

En las pruebas realizadas se detectaron veintinueve no conformidades clasificadas en significativas, no significativas y recomendaciones. Entre las no significativas se detectaron catorce, de las cuales siete fueron errores ortográficos, seis de interfaz y una de apariencia del sistema. Como parte de las no conformidades significativas se encontraron tres de funciones incorrectas y siete errores de validación para un total de diez. Todas quedaron resueltas al finalizar la última iteración. En el siguiente gráfico muestra un resumen de lo explicado anteriormente.

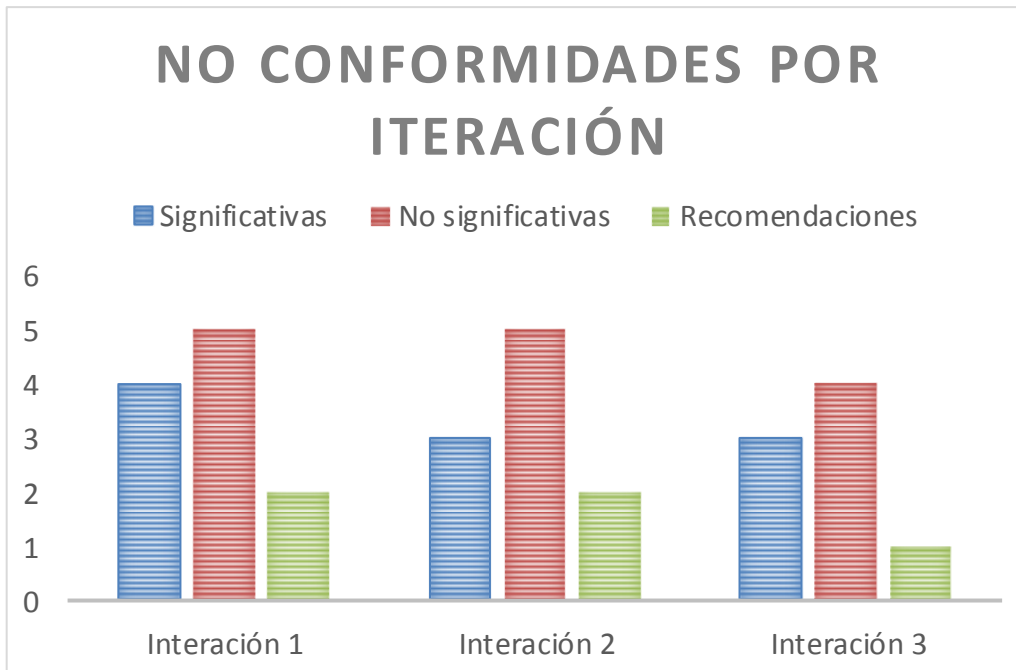


Figura 16: No conformidades detectadas por iteración.

3.4.2 Pruebas unitarias

Para las pruebas unitarias realizadas al código se emplearon los pasos que definen el *framework* Symfony y la librería PHPUnit. A continuación se presentan algunas imágenes tomadas durante la realización de estas pruebas:



Figura 17: Estructura donde se guardan las pruebas a las entidades.

Las pruebas realizadas permitieron validar el código implementado detectándose y corrigiéndose un total de 17 errores distribuidos como se muestra a continuación:

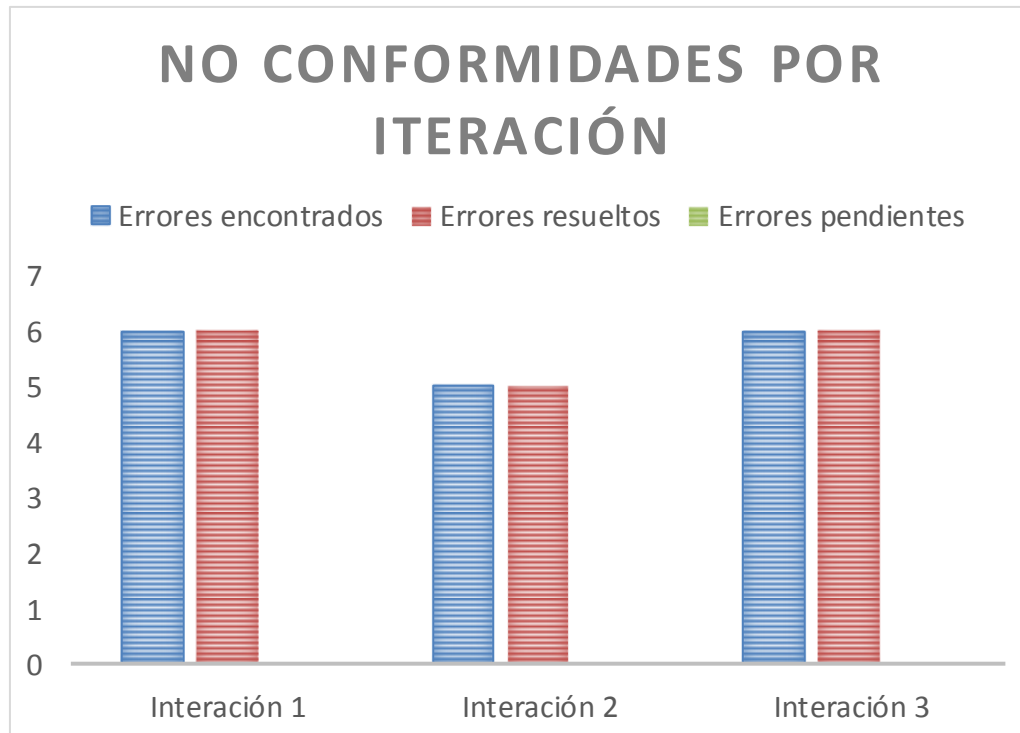


Figura 20: Errores detectados por iteración.

Conclusiones parciales del capítulo

En el presente capítulo se definió el uso de un estándar de codificación para mejorar dicho proceso y facilitar el mantenimiento del código.

El desglose de las historias de usuario en tareas de ingeniería facilitó la implementación y permitió a los programadores obtener en cada iteración, una versión funcional del producto.

Los métodos de pruebas aplicados permitieron validar el correcto funcionamiento de la aplicación, tanto a nivel de código como en la interacción con el usuario comprobándose que las funcionalidades son las esperadas por el cliente.

Conclusiones

Después de desarrollar la propuesta de solución y analizar los resultados obtenidos, se arriba a las siguientes conclusiones:

- Luego de haber realizado el estudio para dar solución al problema planteado en la presente investigación se obtuvo como resultado la necesidad de la elaboración de una herramienta que permita gestionar el proceso de revisiones técnicas formales en el área de Arquitectura de desarrollo del software en el centro FORTES.
- La selección de la metodología de desarrollo, las herramientas, lenguajes de programación y tecnologías más apropiadas para dar cumplimiento al objetivo general constituye un resultado del análisis del marco teórico realizado.
- El diseño desarrollado para la aplicación, permitió la implementación de funcionalidades que dieron solución a la problemática descrita.
- Los métodos de pruebas aplicados permitieron validar el correcto funcionamiento de la aplicación, tanto a nivel de código como en la interacción con el usuario comprobándose que las funcionalidades son las esperadas por el cliente.

Recomendaciones

- Continuar con el estudio del proceso de revisiones técnicas formales en el área de Arquitectura de desarrollo del software con el propósito de obtener mayores conocimientos e identificar nuevos requisitos para actualizar la aplicación obtenida, garantizando mayor funcionalidad y aceptación por parte del cliente.
- Implantar la solución como parte del proceso de desarrollo de software del centro FORTES.
- Agregar funcionalidades al sistema que permitan una mayor usabilidad del mismo.

Bibliografía

- Metodología X. 2014. [En línea] 2014. [Citado el: 9 de Abril de 2015.]
<https://sites.google.com/site/xpmetodologia/marco-teorico/funcionamiento>.
- ProcesosdeSoftware. 2014. METODOLOGIA XP. [En línea]
<http://procesosdesoftware.wikispaces.com/METODOLOGIA+XP>), 2014. [Citado el: 25 de Marzo de 2015.] <http://procesosdesoftware.wikispaces.com/METODOLOGIA+XP>.
- Abreu, Ing. Rogelio Barroso, y otros. 2008. *Aplicación de la metodología ágil Crystal Clear a un caso de estudio*. 2008.
- Acuña, E. A. 2001. *Calidad del proceso*. 2001.
- Alfaro, W. V. 2008. *Simulación de proceso*. 2008.
- Álvarez, M. A. 2004. *Introducción a PHP 5*. 2004.
- Alvarez, Miguel Angel. 2011. *Manual de jQuery*. 2011.
- Alvarez, Miguel Angel, y otros. 2013. *Manual de CSS, hojas de estilo*. 2013.
- Álvarez, R. S. 2008. *Aseguramiento de calidad de software*. 2008.
- Antonio, A. D. 2008. *Gestión, control y garantía de la calidad de software*. 2008.
- Aranda, Vicente Trigo. 2001. *Historia y evolución de los lenguajes de programación*. 2001.
- Ariel Erlijman Piwen, Alejandro Goyén Fros. 2001. *Problemas y Soluciones en la implementación de Extreme Programming*. 2001.
- Bass, Clements, y Kazman. 1998. *Software Architecture in Practice*. 1998.
- BAUKES, Mike. 2013. ScriptRock. MySQL vs Postgres. [En línea] 9 de Agosto de 2013. [Citado el: 16 de 1 de 2015.] <https://www.scriptrock.com/articles/postgres-vsmysql/>.
- Bertino, E. A. y MARTINO, L. A. 1995. *Sistemas de bases de datos orientadas a objetos*. 1995.
- Bichachi, Diana Susana. 2010. *EL USO DE LAS LISTAS DE CHEQUEO (CHEK -LIST) COMO HERRAMIENTA PARA CONTROLAR LA CALIDAD DE LAS LEYES*. 2010.

Canós, Joseph, Letelier, Patricio y Penadés, M^a Carmen. 2003. *Metodologías Ágiles en el desarrollo de Software*. Valencia: s.n., 2003.

Carmona, Luz Elena Delgado y Tobón, Luis Miguel Echeverry. 2007. *Caso práctico de la metodología ágil XP al desarrollo de software*. Universidad Tecnológica de Pereira: s.n., 2007.

Carrasco, Oscar M. Fernández y León, Delba García. 1995. *ACIMED*. Ciudad de La Habana: s.n., 1995. ISSN 1024-9435.

Casal, Martín. 1989. *La Técnica de las Cheskliten*. 1989.

Centro Nacional de Tecnologías de Información. 2012. <http://merinde.net>. [En línea] 2012.

[Citado el: 2015 de Abril de 9.]

http://merinde.net/index.php?option=com_content&task=view&id=490&Itemid=291.

Check list / Listas de chequeo: ¿Qué es un checklist y cómo usarlo?

Cochran, David. 2012. *Bootstrap Web Development (1st edición)*. . 2012.

Cornejo, José Enrique González. 2005. *¿Qué es UML?* 2005.

DANNY. 2008. Techlosofy. [En línea] 2008. [Citado el: 16 de 1 de 2015.]

<http://techlosofy.com/que-es-ajax..>

Díaz, Ernesto Vladimir Pereda y Pons, Yaismel Miranda. *Pautas de codificación*. .

—. 2012. *Pautas de codificación*. . 2012.

Doctrine Project Team. 2011. *Doctrine-2-Orm*. . 2011.

Donatien, Ariagna Rodriguez. 2011. *Descripción de la Metodología de Desarrollo de Software Agile Unified Process (AUP)*. 2011.

Dorado, M. Domínguez. 2005. *NetBeans IDE 4.1. La alternativa a Eclipse*. Madrid: Iberprensa, 2005.

—. 2005. *Todo Programación. Nº 13. Págs. 32-34. Noviembre. NetBeans IDE 4.1. La alternativa a Eclipse*. Madrid: Editorial Iberprensa, 2005. DL M-13679-2004.

2014. Editores de Código. [En línea] Junio de 2014. <http://www.editoresdecodigo.com/2014/06/>.

El papel de la Arquitectura de Software en Scrum. Mago, Edwin Rafael y Alférez, Germán Harvey. 2010. 2010.

Enriquez, Osmel Yanes y Busto, Hansel Gracia del. 2011. *Mapeo Objeto / Relacional (ORM)*. 2011.

2013. Extensión de la herramienta Visual Paradigm para la generación. [aut. libro] Michael Eduardo Marreno, Yanet Rosales, Yanet Morales, Clark, Oliva. Adrián Trujillo. *Extensión de la herramienta Visual Paradigm para la generación de clases de acceso a datos con Doctrine 2.0*. 2013, Vol. 6.

Fuentes, Juan Mariano. 200. *Manual de AJAX. Las entrañas de AJAX*. 200.

Gutierrez, Javier. 2010. *¿Qué es un framework web?* 2010.

Institute of Electrical and Electronics Engineers. 1990. *Standard Glossary of Software Engineering Terminology*. 1990.

Instituto Tecnológico de Veracruz. 2007. *Perspectiva histórica del Internet*. 2007.

J. J. Gutiérrez, M. J. Escalona, M. Mejías, J. Torres. 2010. *PRUEBAS DEL SISTEMA EN PROGRAMACIÓN EXTREMA*. Sevilla: s.n., 2010.

La Tecnología de la Ingeniería Inversa: Un Método con UML, guiado por Casos de Uso y basado en el Modelo de Vistas 4+1. Diana PALLIOTO, Gabriel ROMANO.

Lazo, Rene. 2011. *Tesis de maestría Modelo de referencia para el desarrollo arquitectónico de sistemas de software en dominios de gestión*. 2011.

León, Eduardo. 2000. *Tutorial Visual Paradigm for UML*. . 2000.

Letelier, Patricio y Penadés, M^a Carmen. 2013. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)*. Valencia: Universidad Politécnica de Valencia, 2013.

Lianet González, Lianet Cabrera and TORRES, Enrique Roberto Pompa. 2012. Extensión de Visual Paradigm for UML para el desarrollo dirigido por modelos. 2012, Vol. 5.

Lisandra Guilbert Estrada, Enrique José Altuna. *INGENIERÍA DE REQUISITOS DEL SOFTWARE EDUCATIVO*. s.l. : Universidad de Ciencias Informáticas(UCI).

Listas de Chequeo. PHILIPS. 2013. s.l. : Voltimum Colombia, 2013.

- Lizbeth A.Hernández González, Juan.Fernández Peña. *Herramienta para la Automatización de Revisiones Técnicas Formales como apoyo al desarrollo Orientado a Objetos.RETO*. Facultad de Estadística e Informática, Universidad Veracruzana.
- Lomprey, Gérald y Hernandez, Saulo. 2008. *LA IMPORTANCIA DE LA CALIDAD EN EL DESARROLLO DE*. México: Facultad de Ingeniería y Tecnología, Universidad de, 2008.
- Markiewicz, Marcus Eduardo y Carlos, J.P. de Lucena. 2001. *El Desarrollo del Framework Orientado al Objeto*. . 2001.
- Marqués, Asier. 2012. *Desarrollando un framework REST sobre los componentes de Symfony2*. 15-16 de Junio de 2012.
- Marset, Rafael Navarro. 2006-2007. REST vs Web Services. Modelado, Diseño e Implementación de Servicios Web. [En línea] 2006-2007. [Citado el: 2015 de 1 de 20.]
- Martínez, Vicen. 2013. Desarrollo web y Marketing online. Introducción a Symfony2. [En línea] 24 de Abril de 2013. [Citado el: 19 de 1 de 2015.] <http://vicenmartinez.wordpress.com/2013/04/24/introduccion-a-symfony2/>.
- Master de Desarrollo Local. 2010. *Cuadro de Mando Integral*. 2010.
- Mella, MSC. Patricio Fabián Oliva. 2009. *Construcción de listas de chequeo*. 2009.
- Menéndez, Anniel Arencibia Morales y Milena Sánchez. 2012. *Módulo de tarjeta control para el sistema synta*. 2012. Vol. 5.
- Mestras, Juan Pavón. 2009. *Estructura de las Aplicaciones Orientadas a Objetos El patrón Modelo-Vista-Controlador (MVC)*. 2009.
- Mir, Josep Huguet. 2012. Estudio de los futuros estándares html5 y css3. Propuesta de actualización del sitio www.mpiua.net. [En línea] Universidad de Lleida, Septiembre de 2012. [Citado el: 2015 de 1 de 15.]
- Mora, Sergio Luján. 2002. *Programación de aplicaciones web: historia, principios básicos y clientes web*. 2002.
- MORÁGUEZ I., A. 2005. *Curso de estadística aplicada a la investigación educacional. Materiales impresos, compendio de tablas y ejercicios adaptados para el curso*. . Holguín : s.n., 2005.

NASA-GB-A302, OFFICE OF SAFETY AND MISSION ASSURANCE. 1993. [En línea] August de 1993. <http://www.everyspec.com>.

NU. CEPAL (Comisión Europea). 2009. Desafíos y oportunidades de la industria del software en América Latina. *Desafíos y oportunidades de la industria del software en América Latina*. Vitacura: CEPAL, Mayol Ediciones, 2009, Vol. 3.

Osvaldo Díaz Verdecia, Virgen Damaris Quevedo Campins. 2009. *Una guía práctica de Arquitectura de Software*. La Lisa: s.n., 2009.

Palacio, Juan. 2006. *El modelo Scrum*. 2006.

Paniagua, Angel Barbero. 1999. *Tutorial de XML*. 1999.

Patricio, Penadés LETELIER, M^a Carmen Penadés. 2006. *Metodologías ágiles para el desarrollo: eXtreme Programming (XP)*. 2006. Vol. 5.

Pecos, Daniel. 2008. *PostgreSQL vs. MySQL*. . 2008.

Pérez, Javier Eguíluz. 2008. *Introducción a AJAX*. 2008.

Phillipe Krutchen, Grady Booch, James Rumbaugh. 2008. Vista de procesos. [En línea] 20 de Octubre de 2008. <http://clases3gingsof.wikifoundry.com/page/Vista+de+procesos>.

Piccolo, Rafael. 2006. *Estandares e indicadores de calidad*. 2006 .

POTENCIER, Fabien. 2011. What is Symfony2? . [En línea] 25 de November de 2011. [Citado el: 19 de 1 de 2015.] <http://www.fabien.potencier.org/article/49/what-is-symfony2>.

Potencier, Fabien y Zaninotto, François. 2008. *La Guía Definitiva de Symfony*. 2008.

Pressman, Roger. 202. *Ingeniería del Software. Un enfoque práctico*. 5ta. 202.

Pressman, Roger S. 2005. *Ingeniería del software: un enfoque práctico*. 2005.

Prieto, Félix. 2009. *Patrones de diseño*. . 2009.

Quintero, Juan Bernardo, y otros. 2012. *Un estudio comparativo de herramientas para el modelado con UML*. 2012. Universidad EAFIT, Vol. 41, págs. 60-76.

Reinoso, Carlos Billy. 2014. *Introducción a la arquitectura de software*. Buenos Aires : s.n., 2014.

Responsive CSS Framework Comparison. Bootstrap vs. Foundation vs. Skeleton. VERMILION. 2013. 21 de November de 2013.

Ruiz, Yosvany Márquez. 2014. *Lista de chequeo para evaluar la Arquitectura de Software.* 2014.

—. 2014. *0120_Arquitectura de Software Vista_de_Presentacion.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Arquitectura_de_Software Guía base.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Arquitectura_Vista_de_Entorno_de Desarrollo_Tecnologico.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Arquitectura_vista_de_infraestructura.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Arquitectura_vista_de_seguridad.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Arquitectura_vista_de_sistema.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Arquitectura_viste_de_datos.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Vista_de_despliegue.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *0120_Vista_de_integracion.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

—. 2014. *Lista de chequeo para evaluar la Arquitectura de Software.* s.l. : Expediente de Arquitectura.(Universidad de las Ciencias Informáticas), 2014.

SALVADOR BROCHE, Orlando Felipe. 2013. *Indicaciones para el trabajo en el marco de trabajo Xalix.* La Habana: Universidad de las Ciencias Informáticas, 2013.

Sánchez, Jorge. 2011. *Servidores de Aplicaciones Web.* 2011.

slideshare.net. [En línea] [Citado el: 18 de 1 de 2015.] http://es.slideshare.net/landeta_p/2-1-vistas-arquitectonicas.

Sommerville, Ian. 2011. *Software engineering*. Boston : s.n., 2011. ISBN 0-13-705346-0.

Sotés, MsC. Roberto Félix Zamuriano. 2012. *Las Inspecciones de Software y las Listas de Comprobación*. 2012.

Taft, Darryl K. 2013. *JetBrains PhpStorm 5.0 Provides New PHP Framework Support*. 2013.

The Apache Software Foundation. 2012. *Apache License and Distribution*. 2012.

Umbrello UML Modeller Autores. 2003. *Manual de Umbrello UML Modeller*. 2003.

Un estudio comparativo de herramientas para el modelado con UML. Bernardo Quintero, Juan, Anaya de Páez, Raquel, Marín, Juan Carlos y Bilbao López, Alex. 137, Universidad EAFIT, Vol. 41, págs. 60-76.

Universidad de Morón. 2011. *Herramientas de Software*. 2011.

Verdecia, Osvaldo Díaz, y otros. 2010. *Propuesta de vistas de la arquitectura de software*. 2010.

Yonvanny, Benigni, Gladys, Antonelli, Octavio y Vázquez. 2009. *Ciencias Básicas y Tecnología*. 2009. págs. 60-69. Vol. 21.