Facultad 4

Informatización del procedimiento de Revisiones Técnicas Formales en el área de Implementación del desarrollo de *software* para el centro FORTES

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

Autores:

Daynier Guerrero Chacón

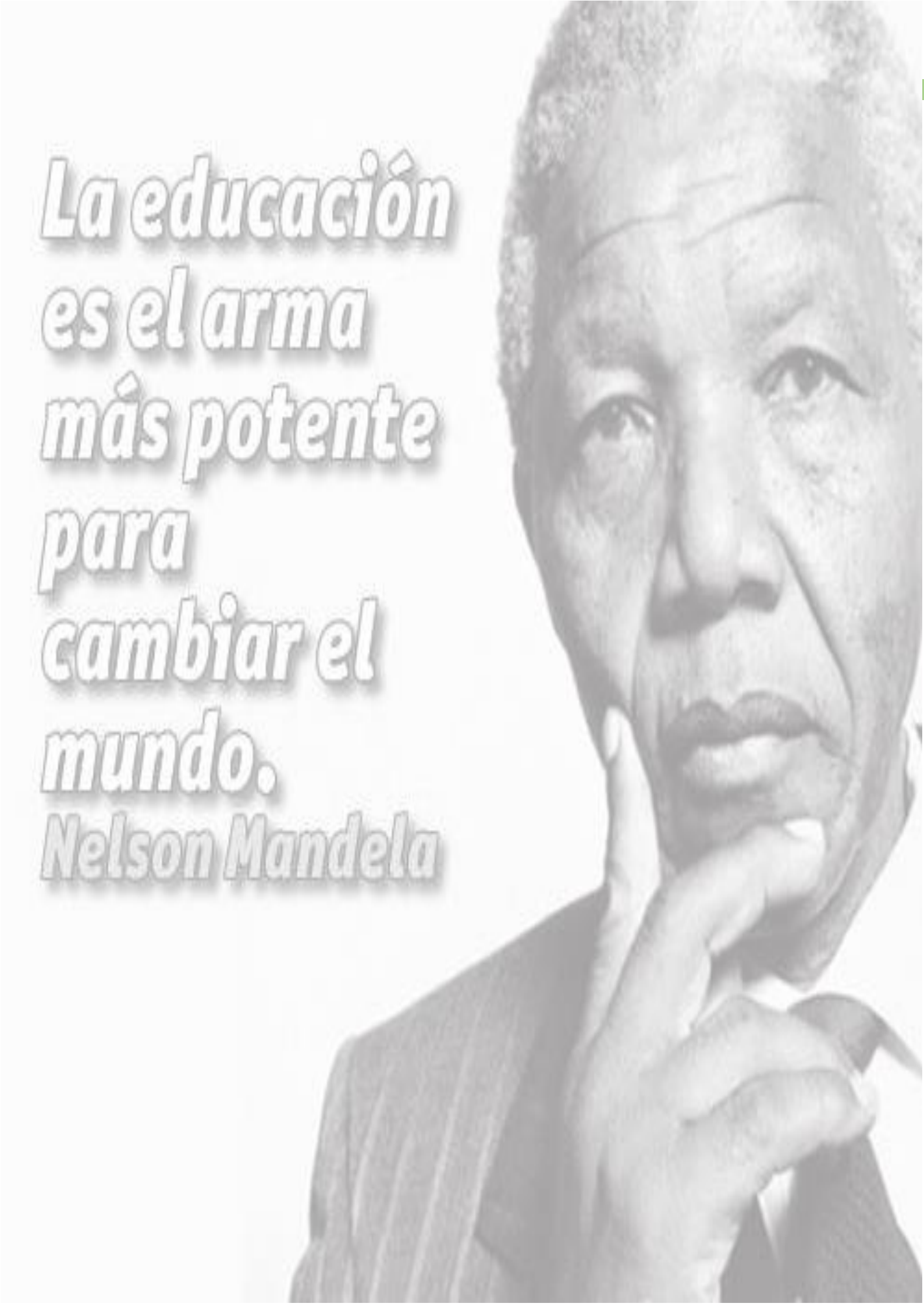
Eddy Rubén Rodríguez Expósito

Tutores:

Ing. Lizardo Ramírez Taboada

Ing. Leannys Rodriguez Moreno

La Habana, junio 2015


La educación
es el arma
más potente
para
cambiar el
mundo.
Nelson Mandela

DECLARACIÓN DE AUTORÍA

Por este medio declaramos ser autores del presente Trabajo de Diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de junio del año 2015.

Autores

Daynier Guerrero Chacón	Eddy Rubén Rodríguez Expósito

Tutores

Ing. Lizardo Ramírez Tabuada	Ing. Leannys Rodriguez Moreno

Agradecimientos de Eddy Rubén

Agradezco a todos los que de una forma u otra me han apoyado en lograr graduarme en esta Universidad. Son muchas las personas especiales que conocemos, intentando mencionarlas a todas se corre el riesgo de olvidar a alguna. Entre todas esas personas hay tres agradecimientos especiales, uno de ellos para mi madre Julia, como todos los hijos decimos que la mejor madre del mundo es la nuestra, pero realmente esta es la mejor del mundo, mucho de lo que soy se lo debo a ella, es especial en todos los sentidos. El otro agradecimiento especial es para mis dos abuelos Rubén y Mariel, los mejores abuelos del mundo, abuelos incomparablemente buenos, mi abuelo es como mi padre y mi abuela como mi madre de principios. EL último agradecimiento especial es para el país donde vivo, este país que con sus grandes dificultades sigue siendo maravilloso y puede serlo más aún si trabajamos juntos convencidos que puede convertirse en un hermoso lugar donde vivir.

A todo mis compañeros, todos los que he conocido de los ¨más oscuros rincones de esta isla¨ desde Maisí hasta María la Gorda, todos tienen algo especial, muchos quedaron en el camino hacia este día el de su discusión de tesis, hoy no están aquí con nosotros pero estuvieron en un momento y forman parte por ello de esta linda historia.

A mis tutores Leannys y Lizardo por todo el apoyo brindado y a mi compañero de tesis, al final parece que salimos airosos después de tanto esfuerzo.

A mi familia ¨Los Expósitos¨ que espera por fin su primer ingeniero y por ello me ha apoyado de muchas maneras, todos los primos, tíos, hermano, al fin voy a poder estar con ustedes a tiempo completo.....se acabaron las becas.....

Agradecimientos de Daynier

A mis padres, pues a ellos les debo quien soy, gracias por el apoyo y el amor brindado y por haber creído siempre en mí.

A mi familia por estar ahí en todo momento para ayudarme a seguir adelante, en especial a mi primo Yander.

A mi novia por su amor, respeto, comprensión y paciencia.

A mis tutores Lizardo y Leannys por su apoyo, dedicación, paciencia y entrega.

A mis amigos Neysa, Angel Luis, Eddy por los momentos buenos y malos que hemos compartido en este tren que hemos viajado durante cinco años.

Y a todos con los que he tenido la oportunidad de compartir en esta universidad, mis compañeros de grupo en los cuales he tenido buena relación tanto en tareas docentes como de otro tipo.

Resumen

El presente trabajo surge con la necesidad de agilizar el procedimiento de Revisiones Técnicas Formales en el área de Implementación del Centro de Tecnologías para la Formación de la Universidad de las Ciencias Informáticas. Este procedimiento actualmente no es realizado en el centro por lo que varios errores humanos podrían no ser detectados y esto podría comprometer la calidad de los productos. Para mejorar esta situación se realizó un estudio referente a este tipo de revisiones tanto de forma general como específicamente en el área de Implementación. Además se estudiaron conceptos relacionados con la calidad de *software* y de esta forma se arribó a conclusiones con el objetivo de realizar el proceso de revisiones de forma eficaz. Posteriormente se realizó un estudio de diferentes herramientas similares, además de un conjunto de tecnologías y herramientas con el fin de desarrollar la propuesta de solución. Luego se diseñó, implementó y se le realizaron pruebas a la herramienta. Esta facilita la realización de las revisiones y ayuda a los especialistas a desarrollar este proceso con mayor rapidez y calidad.

Palabras clave: calidad, implementación, Revisiones Técnicas Formales, *software*.

Índice

Índice de tablas

Introducción

Desde el surgimiento de la humanidad y con el desarrollo continuo de su pensamiento y capacidad de creación, el hombre se ha interesado por el perfeccionamiento de lo que crea, la forma en que aprecia dicha perfección ha ido evolucionando y se ha desarrollado con la aparición de las Tecnologías de la Informática y las Comunicaciones (TIC). En la actualidad estas mantienen un desarrollo exponencial, su presencia y utilidad se refleja en las diversas esferas del mundo contemporáneo a través de los diferentes tipos de *software*.

En un mundo cada vez más dependiente de la tecnología, la mayoría de las grandes empresas gestionan sus procesos con un sistema informático, su éxito depende fundamentalmente de la calidad de este. Esta situación plantea la necesidad de disponer de un producto con calidad, que funcione según lo previsto y que esté libre de vulnerabilidades. La calidad, en su sentido amplio, es un atributo esencial del producto, ya que su ausencia deriva una pérdida de valor, usuarios insatisfechos y pérdida financiera (1).

Parte de la industria del *software* en la actualidad adolece aún de problemas como la entrega del producto fuera de tiempo, gasto extra en su producción y la reelaboración por mala calidad. La Ingeniería de *Software* surgió por la necesidad de superar la crisis del *software* para evitar los problemas antes enunciados, no obstante continúan siendo muchos los proyectos en esta industria que no llegan a ser efectivos (2) (3).

El bloqueo económico y tecnológico de los Estados Unidos de América impuesto a Cuba durante más de cinco décadas, sumado a otras situaciones adversas, no ha impedido que la nación se encuentre inmersa en todo un proceso gradual de informatización de la sociedad. Para ello, el país ha desarrollado toda una infraestructura que aunque es básica, constituye el primer paso hacia la soñada y necesaria informatización.

La joven industria de *software* cubana tuvo un impulso significativo con la creación de la Universidad de las Ciencias Informáticas (UCI). Esta universidad constituye uno de los pilares fundamentales en el proceso de informatización y se ha convertido en una de las instituciones educativas que más aporta en la esfera de las exportaciones de productos informáticos en el país. Tiene como tareas fundamentales la formación de profesionales integrales de las ciencias informáticas, así como la producción de *software*, tanto para el consumo nacional como para la exportación.

La UCI cuenta con varios centros de producción de *software* entre los que se encuentra el Centro de Tecnologías para la Formación (FORTES). Este centro brinda diferentes servicios a los proyectos productivos tales como revisiones, auditorías y pruebas de *software* en las diferentes etapas de desarrollo. Además, brinda capacitación y orientación al equipo de desarrollo con el objetivo de garantizar la calidad de sus productos.

Dentro de las revisiones que se ejecutan en el centro se encuentran las Revisiones Técnicas Formales (RTF), estas tienen como objetivo asegurar la calidad del producto que se desarrolla. Se ejecutan en las áreas de Requerimiento, Arquitectura, Implementación y Base de datos y permiten que los errores no sean arrastrados a las fases superiores del desarrollo (4).

El hecho de no realizar este tipo de revisiones en la fase de Implementación podría traer consigo que deficiencias como: el uso inadecuado de las convenciones del lenguaje, errores mecanográficos, incorrecta traducción del diseño, mala utilización de los estándares de codificación para el tipo de lenguaje, comentarios ambiguos, fuera de lugar o incorrectos y violación de patrones, estén presentes en los productos que se desarrollan. Como estas revisiones no se realizan en el centro FORTES no se cuenta con un historial de problemas detectados, por lo que no se puede dar continuidad a defectos anteriores. Tampoco se puede realizar un estudio de tendencias ni contar con un registro de lecciones aprendidas y el proceso de desarrollo se ve entorpecido porque los participantes del proyecto no pueden aprender de los errores cometidos en el pasado.

Debido a la situación antes expuesta se tiene como **problema a resolver**: ¿cómo informatizar el procedimiento de Revisiones Técnicas Formales en el área de Implementación del desarrollo de *software* para el centro FORTES?

El **objeto de estudio** en el cual se centrará la presente investigación es: el procedimiento de Revisiones Técnicas Formales en el desarrollo de *software*.

Para dar cumplimiento al problema planteado se establece como **objetivo general**: informatizar el procedimiento de Revisiones Técnicas Formales en el área Implementación del desarrollo de *software* para el centro FORTES.

Por tanto se definen los siguientes **objetivos específicos**:

- Investigar el proceso de Revisiones Técnicas Formales, conceptos fundamentales, artefactos y diversas fuentes que analizan los procedimientos en la Implementación del desarrollo de *software.*

- Diseñar una herramienta para la informatización del procedimiento de las Revisiones Técnicas Formales en el área de Implementación del desarrollo de *software*.
- Realizar la implementación y pruebas de la herramienta para las Revisiones Técnicas Formales en el área Implementación del desarrollo de *software*.

El **campo de acción** lo constituye: la informatización del procedimiento de Revisiones Técnicas Formales en el área de Implementación del desarrollo de *software* para el centro FORTES.

En correspondencia con el problema a resolver, el objeto de estudio, el objetivo general, los objetivos específicos y el campo de acción, se determinaron las siguientes **preguntas científicas**:

1. ¿Qué fundamentos teóricos, metodológicos y tecnológicos sustentan las tendencias actuales del estudio de las Revisiones Técnicas Formales en el área de Implementación de desarrollo de *software*?
2. ¿Cuál es el estado actual del procedimiento de las Revisiones Técnicas Formales, en el área Implementación del desarrollo de *software* en el centro FORTES?
3. ¿Qué tecnologías se deben tener presente en el diseño de una herramienta para informatizar el procedimiento de Revisiones Técnicas Formales en el área de Implementación del desarrollo de *software* en el centro FORTES?
4. ¿Qué resultados se alcanzan con la implementación y la aplicación de las pruebas de *software* a la herramienta de informatización del procedimiento de Revisiones Técnicas Formales en el área de Implementación en el centro FORTES?

Para garantizar el cumplimiento de los objetivos específicos definidos anteriormente se plantean las siguientes **tareas de investigación**:

- Identificación y análisis de las tendencias actuales sobre las Revisiones Técnicas Formales en el área de Implementación.
- Selección de la metodología que guiará el proceso de desarrollo del *software*.
- Selección de las tecnologías y herramientas informáticas a utilizar en el desarrollo de la herramienta.
- Descripción de las funcionalidades y características de la propuesta de solución.
- Diseño de la propuesta de solución utilizando la metodología seleccionada.
- Implementación y pruebas de la propuesta de solución.

Métodos científicos de investigación

La realización de la investigación implica el uso de los métodos de investigación científica los cuales se dividen en dos grandes grupos, los métodos teóricos y los empíricos, estos permiten descubrir la esencia de los procesos sobre los cuales se desarrollan y de esta manera emitir criterios válidos sobre la exactitud y veracidad de las conclusiones en la investigación.

Métodos teóricos: estos métodos permiten estudiar objetos que no se observan directamente, en la investigación (5).

- Histórico-lógico: fue utilizado para analizar las diferentes aplicaciones informáticas existentes en la actualidad que revisan código de manera automática. Logrando entender las tendencias actuales de este tipo de aplicaciones, funcionalidades principales, carencias y potencialidades.
- Analítico-sintético: utilizado para realizar un análisis de los conceptos fundamentales relacionados con la calidad y de esta forma realizar una profundización de los mismos y arribar a conclusiones lo cual contribuye a una mejor elaboración de la propuesta de solución.

Métodos empíricos: estos se basan en la interacción directa entre el sujeto y el objeto sobre el cual se ejecuta la investigación, se utilizan los siguientes (5):

- Observación: utilizado para identificar las principales actividades de los programadores del centro FORTES cuando desarrollan en sus respectivos proyectos y las acciones de los probadores del centro cuando realizan las pruebas.
- Entrevista: este fue ejecutado con el propósito de obtener información sobre la existencia o no de las RTF en la UCI, para conocer el lugar de ubicación del estándar de codificación definido en cada proyecto productivo, expediente de proyecto en su última versión y con esto los documentos a entregar en la fase de Implementación. También para conocer cómo se desarrolla el proceso de desarrollo en el centro FORTES (ver anexo 6).

Esta investigación cuenta con tres capítulos los cuales están estructurados de la siguiente forma:

Capítulo 1: En este se abordan los conceptos fundamentales relacionados con el problema a resolver. Estudio de la metodología de desarrollo, las herramientas y tecnologías fundamentales a utilizar, su análisis y fundamentación de su selección.

Capítulo 2: En este capítulo se hace una presentación de la propuesta de solución a la problemática. Se realiza un recorrido por las diferentes fases de la metodología de desarrollo a utilizar y en cada una de ellas se presenta el trabajo realizado.

Capítulo 3: En este capítulo se realiza una descripción de todos los elementos que se tuvieron en cuenta durante el desarrollo de la herramienta para informatizar el proceso de RTF en el área de Implementación. Se muestran también, los métodos de prueba aplicados al sistema y el diseño de casos de pruebas correspondientes a cada caso de uso. Por último se muestran los resultados de las pruebas aplicadas a la herramienta.

Capítulo 1. Fundamentación teórica

Introducción

En este capítulo se expone un estudio de los conceptos fundamentales que se deben tener en cuenta para la comprensión del problema a resolver, se hace énfasis por su importancia para la solución del mismo en las revisiones técnicas formales, en especial a las que se realizan a la implementación de *software.* Se mencionan además una serie de herramientas de revisión de código de manera automática. Se determina la metodología de desarrollo de *software*, las herramientas y tecnologías a utilizar.

1.1 Conceptos y elementos fundamentales para la comprensión del problema

1.1.1 Calidad de software

El aseguramiento de la calidad asociado a la producción de *software* es fundamental, determina la eficacia del proceso de ingeniería que se desarrolla en toda producción de este tipo. Son varias las definiciones que existen en el mundo que se refieren a la calidad de *software*, tales como:

IEEE[1] estándar 610-1900 *"la calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario"* (6).

La norma ISO[2] 8402:1994 plantea que *es "el conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas"* (7).

Roger Pressman[3] define que es *"la concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario"* (8).

De las definiciones enunciadas se considera la más apropiada la dada por la IEEE pues es concisa cuando se refiere al cumplimiento de los requerimientos y las necesidades reales del cliente. Además, es la que se relaciona con la calidad que se busca obtener al realizar una RTF.

[1] (*Institute of Electrical and Electronics Engineers*) en español Instituto de Ingeniería Eléctrica y Electrónica.
[2](*International Organization for Standardization*) en español Organización Internacional de Normalización.
[3] Ingeniero de *Software* estadounidense graduado de la Universidad de Connecticut, autor del libro Ingeniería de *Software*, un enfoque práctico.

1.1.2 Revisiones de software

Las revisiones de *software* son una forma de garantizar la calidad del producto que se desarrolla. Se aplican en cada una de las fases de ingeniería de *software* y sirven para descubrir errores a tiempo, de no ser así luego pueden lastrar la calidad del producto (4).

Las revisiones del *software*, son el conjunto de actividades que suceden como resultado del análisis, el diseño y la codificación y que sirven para depurar las actividades de ingeniería del *software* (9).

Tiene como objetivos (9):

- Señalar la necesidad de mejoras en el producto.
- Continuar las partes de un producto en las que no es necesaria o no es deseable una mejora.
- Conseguir un trabajo técnico de una calidad más uniforme o al menos más predecible, que la que puede ser conseguida sin revisiones, con el fin de hacer más manejable el trabajo técnico.

1.1.3 Revisiones Técnicas Formales

Las RTF son revisiones que se realizan a través del proceso de ingeniería de *software* verificando que cada una de las etapas que lo componen se desarrolle con la calidad requerida, su realización con rigor y de manera organizada constituye la forma más efectiva de no arrastrar errores a etapas superiores. Contribuye de forma significativa a la calidad final que debe caracterizar el producto, garantiza la satisfacción del cliente y trae como resultado que la institución desarrolladora gane en prestigio (4).

Objetivos de las RTF (4)

1. Descubrir errores en la función lógica o implementación en cualquier representación del *software*.
2. Verificar que el *software* en revisión satisface sus requisitos.
3. Garantizar que el *software* se ha representado de acuerdo con los estándares predefinidos.
4. Lograr un desarrollo de *software* uniforme.
5. Hacer proyectos manejables.

1.1.4 Revisiones Técnicas Formales a la Implementación

Importancia

La automatización en la generación de código es clave para conseguir un incremento en la calidad y

productividad del desarrollo de *software*. Sin embargo, todavía estamos lejos de una automatización total y sigue siendo necesario manipular el código más de lo deseable. La buena calidad del código propiciada por el estricto apego a estándares de codificación y patrones de diseño, favorece el rendimiento de la aplicación, se hace fácil de leer por otros desarrolladores y por tanto sencillo de mantener. Un código que por el contrario tenga un diseño deficiente, que presente por ejemplo una funcionalidad duplicada en diferentes partes del mismo, es difícil de mantener. Un código de fácil mantenimiento permite añadir de forma más sencilla nuevas características al sistema, así como modificar las existentes, depurar errores o mejorar el rendimiento con mayor facilidad (10) (11).

Para tener una noción del impacto de corregir un error en la misma fase donde se originó se tiene el siguiente estudio: en cinco minutos se puede corregir una deficiencia detectada durante el desarrollo, mientras que corregir la misma durante el *testing* funcional puede llegar a demandar treinta minutos (contando el tiempo requerido para reproducirla y repararla). En el peor de los casos, si esta es detectada por un usuario final, corregirla puede demorar hasta una hora (sumando el tiempo que se requiere del usuario, del funcional y del desarrollador). Este estudio demuestra cuán importante es corregir un defecto en la fase que se originó (12).

Elementos a revisar

El código fuente y la documentación generada durante la fase de Implementación son los candidatos para las inspecciones. Estas deben comprobar la precisión técnica y la integridad del código, verificar que se implementó lo que se planificó en la etapa de diseño y asegurar las buenas prácticas de codificación, así como comprobar si se utilizaron los estándares definidos previamente. Estas se deben hacer después que el código ha sido compilado y todos los errores de sintaxis hayan sido eliminados. También, se revisan los documentos generados en la fase, dígase Manual de usuario y Guía de instalación y configuración presentes en el expediente de cada proyecto (13).

Un Manual de usuario es una publicación que brinda las instrucciones necesarias para que un usuario pueda utilizar un determinado producto o servicio. Es un documento de comunicación técnica que busca brindar asistencia a los sujetos que usan un sistema. Más allá de su especificidad, los autores de los manuales intentan apelar a un lenguaje ameno y simple para llegar a la mayor cantidad posible de receptores. Al realizar la RTF se debe verificar que este cuente con la estructura correspondiente así como que no tenga errores ortográficos (14).

El código fuente de una aplicación es texto escrito en un lenguaje de programación específico y que puede ser leído por un programador. Debe traducirse a un lenguaje de máquina para que pueda ser ejecutado por la computadora o a *bytecode* para que pueda ser ejecutado por un intérprete (15).

1.1.5 Lenguaje de programación

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina (16).

Los siguientes son los paradigmas en los que se puede clasificar (17):

- Paradigma imperativo.
- Paradigma funcional.
- Paradigma lógico.
- Paradigma orientado a objetos (POO).
- Paradigma estructurado.

El POO es de los más utilizados en la actualidad por sus ventajas entre las que se pueden mencionar una fácil ejecución del polimorfismo, herencia, encapsulamiento, abstracción. Además, en ocasiones es utilizado en combinación con otros como el estructurado y el imperativo para el desarrollo de una herramienta informática.

La tabla 1 muestra los lenguajes de programación más utilizados por los desarrolladores del centro FORTES así como los *framework* de desarrollo asociados a cada uno. Cada uno de estos lenguajes de programación tiene definido estándares de codificación, así como buenas prácticas de desarrollo para cada *framework.*

Tabla 1. Lenguajes de programación más utilizados en FORTES

Lenguaje de programación	Framework
Java	Spring MVC, JFS, Vaadin
JavaScript	Dojo Toolkit, ExtJS, JQuery, Mootools
PHP	Symfony, CodeIgniter

1.1.6 Estándar de codificación

Un estándar de codificación es una guía que sirve a los programadores para mantener un mismo estilo en el código desarrollado asegurando de esta forma su legibilidad y mantenimiento. Comprende todos los aspectos de la generación de código. Los programadores deben implementar un estándar de forma prudente que debe tender siempre a lo práctico. Su uso permite que un código fuente completo refleje un estilo armonioso, como si un único programador lo hubiera escrito de una sola vez. Son el mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad (11).

Existen estándares de codificación definidos para los lenguajes de programación más conocidos en el mundo. Por lo general cada institución desarrolladora de *software* define previamente al comienzo de cada proyecto el estándar de codificación a utilizar sobre cada uno de los lenguajes a utilizar en la confección del *software*, no es recomendable introducirlo una vez comenzado el desarrollo del producto (11).

Definen fundamentalmente los siguientes aspectos, para asegurar la estandarización del código fuente que se genere (11):

1. Declaración de variables, tablas, cadenas, listas, pilas y colas.
2. Métodos con sus parámetros.
3. Clases con susparámetros.
4. Operadores.
5. Comentarios.
6. Funciones lógicas.
7. Abreviaturas.
8. Eventos.
9. Excepciones.
10. Enumeración.

Una vez realizado el estudio de estos conceptos podemos llegar a la conclusión de que es de suma importancia que los productos informáticos cuenten con la calidad necesaria. Una forma de contribuir a la misma es la realización de las RTF las cuales deben ser ejecutadas por especialistas con el fin de lograr un *software* de alto nivel. Es importante destacar que las RTF que se realizan en el área de Implementación aseguran que exista una similitud en el código que se escribe por los diferentes desarrolladores, de esta forma es fácil realizarle tareas de mantenimiento. Por último, estas revisiones

a la Implementación aseguran la correcta elaboración de los documentos generados lo que propicia un mejor entendimiento de los usuarios que interactuarán con el *software* que se desarrolle.

1.2 Análisis de procedimientos de revisiones de software

1.2.1 Propuesta del Instituto Superior Politécnico José Antonio Echeverría

El Instituto Superior Politécnico José Antonio Echeverría de la Ciudad de la Habana desarrolló una propuesta de introducción de las revisiones en el proceso desarrollo de *software* para las Pequeñas y Medianas Empresas Productoras de *Software* en Cuba (PYME). Esta consta de tres fases (18):

- Preparación de las revisiones: en esta etapa la empresa deberá trazar pautas para el desarrollo de las revisiones, los roles que se ejecutarán, así como las políticas generales de aseguramiento de calidad.

- Gestión de productos terminados: el objetivo fundamental de este proceso es contar con la información del estado y calidad de los productos desarrollados por la empresa antes de introducir el modelo, de forma que esta pueda trazar acciones preventivas.

- Gestión de nuevos productos y de productos en desarrollo: la organización debe definir un conjunto de políticas con respecto a las revisiones, que deberán ser seguidas por desarrolladores e inspectores a lo largo del proceso de desarrollo de *software* y durante la ejecución de las revisiones. Se definirán los momentos, tipos de revisión y listas de chequeo o listas de comprobación generales para cada tipo de proyecto, que son aquellas listas cuyos elementos consisten en parámetros o preguntas a considerar en la revisión y que pueden ser defectos que están presentes en el proyecto. Además, es importante que no deje de declararse explícitamente con qué frecuencia se revisará la línea base del proyecto que pasará a formar parte del repositorio de elementos de configuración, como una versión estable y revisada de los elementos de configuración correspondientes.

1.2.2 Propuesta de la Universidad de la Ciencias Informáticas

En la UCI existe definido un procedimiento para la realización de las RFT, posibilitando que las mismas sean constantemente examinadas, evaluadas y mejoradas. A continuación se describen en detalles las fases, etapas y actividades correspondientes a la actividad productiva de esta universidad (19):

Inicio de la revisión

En esta fase se analiza la necesidad de realizar una RTF a alguna organización productiva y se selecciona el personal con posibilidad de llevar a cabo las auditorías. Se notifica a los involucrados la realización de la RTF y se designa el revisor líder. Posteriormente se verifica si existe información suficiente y apropiada para planificar la RTF. Finalmente se definen los objetivos de la revisión.

Desarrollo de la RTF

En esta fase como su nombre lo indica es en la que se realizan las RTF. Aquí se preparan las actividades correspondientes, se confirma la disponibilidad de los recursos y se asignan las tareas al equipo revisor. Luego se prepara el plan de revisión y se les asignan las tareas a los revisores. Posteriormente se generan los hallazgos de las revisiones, se realiza un informe preliminar de la RTF y se presentan acciones correctivas o de mejora.

Finalizar la RTF

En esta fase se tramita el informe de la RTF, se prepara el expediente y se almacena la información. Finalmente se evalúa el desempeño de los revisores en las tareas realizadas.

1.2.3 Revisiones Técnicas Formales a la Implementación en el centro FORTES

Actualmente en el procedimiento de RTF llevado por calidad UCI no se cuenta con listas de chequeo ni está concebido el subproceso de revisiones a la fase de Implementación, estas se realizan en los diferentes centros debido a las particularidades de cada uno. Para garantizar la calidad del *software* en el centro FORTES se pretende a través de una herramienta revisar la documentación generada en esta etapa de desarrollo y el código basándose en los estándares de codificación con que se cuenta para el desarrollo de los productos.

Con vista a estandarizar el proceso de las RTF, partiendo de que la bibliografía contempla realizar revisiones a la Implementación, se elaboraron listas de chequeo (ver anexo 5) para revisar la documentación antes mencionada y el código que se genera en esta etapa. Fueron confeccionadas tres listas de chequeo, las mismas están conformadas generalmente por preguntas o por afirmaciones de elementos que no deben faltar tanto en la documentación como a la hora de programar.

A continuación se presentan estas listas con una breve descripción de los principales elementos que miden:

Lista de chequeo para el lenguaje PHP

La lista de chequeo fue elaborada teniendo como base el estándar de codificación definido para este lenguaje en el centro FORTES. Se encarga de verificar la forma en que fue escrito el código fuente en una aplicación desarrollada en el centro. Entre los elementos más destacados que la conforman se encuentra la forma correcta de declaración de las estructuras condicionales, métodos, clases, *namespace* de la clase, sentencias *use* y arreglos multilíneas. La longitud de las líneas de código y la forma en que se escriben los comentarios también se tienen en cuenta.

Lista de chequeo de la Guía de instalación y configuración

La lista de chequeo fue elaborada teniendo como base la plantilla de este documento que se encuentra en el expediente de proyecto que define el centro FORTES. La lista se encarga de verificar la calidad con que fue elaborado este documento, para ello tiene en cuenta la ortografía, la claridad y exactitud en la redacción de las ideas, el cumplimiento de las convenciones tipográficas que están definidas. También tiene en cuenta la calidad y claridad de las imágenes que se utilizan, los nombres de tablas y figuras y la existencia de cada una de las partes imprescindibles que deben estructurar el documento.

Lista de chequeo para el Manual de usuario

La lista de chequeo fue elaborada a partir de la plantilla del Manual de usuario que se encuentra en el expediente de proyecto que define el centro FORTES. La lista se encarga de la ortografía, la claridad y exactitud en la redacción de las ideas, la portada, el cumplimiento de las convenciones tipográficas que están definidas, la calidad y claridad de las imágenes que se utilizan, los nombres de tablas y figuras y la existencia de cada una de las partes imprescindibles que lo componen.

1.3 Herramientas similares

Para la informatización del proceso fueron estudiadas varias herramientas informáticas de revisión de código de manera automática en diferentes lenguajes de programación, con el objetivo de analizar cómo aseguran la calidad en los productos y de esta forma poder tomar ideas para la realización de la herramienta para informatizar el procedimiento de las RTF en el área de Implementación. Las siguientes son ejemplos de las más representativas:

1.3.1 Aplicaciones para evaluación de código

- Code Crawler: Es una herramienta de depuración de posibles vulnerabilidades en el código fuente para código en .NET y J2EE/JAVA. Es una aplicación desarrollada en .Net 3.5, que se basa en una maquinaria muy afinada de control de expresiones regulares, que permitirá encontrar errores tradicionales (20).

- Review Board: Es una herramienta web para la revisión de código Java muy configurable. Cada usuario tiene un Dashboard[4] donde ve las revisiones de entrada y de salida asignadas a él. Para su funcionamiento necesita una base de datos MySQL, PostgreSQL o SQLite (21).

- PMD: Es una herramienta de calidad de código encargada de validar los estándares de construcción de un desarrollo en lenguaje Java. Es decir, chequea la sintaxis del código fuente que ha sido desarrollado, encontrando las ocurrencias de un determinado problema que haya sido previamente configurado para ser detectado (22).

- Sonar: herramienta de evaluación de código Java que entre sus principales funcionalidades están (23):
    1. Descubrir el volumen de comentarios de una aplicación, así como el índice de duplicaciones.
    2. Disponer de medidas de cobertura de las clases de prueba sobre código de la aplicación/componente.
    3. Integra las mejores herramientas de medición de la calidad de código: CPD, Findbugs, PMD, Checkstyle, agregando la información de dichos *plugins* y ofreciéndote un resumen tipo cuadro de mando. Clasifica las incidencias en base a su severidad y a su naturaleza.
    4. Permite navegar en el código y ver los errores dentro de él mismo. Por tanto, la herramienta ayuda a disponer de un entorno de mejora continuo.

Luego de estudio de estas herramientas se puede concluir que son muy eficaces pero no van más allá de la revisión de código, de esta manera quedan desatendidas otras partes del *software* y se compromete su calidad. Algunas incluso carecen de la flexibilidad necesaria para poder adecuar lo que se debe revisar a nuevos parámetros de calidad, esto atenta contra su utilidad como aplicación. Además, no tienen en cuenta la documentación que se genera en la etapa de implementación ni se

[4] Es una interfaz donde el usuario puede administrar o configurar el equipo y/o *software*.

verifica que los documentos cumplan con estándares de calidad necesarios. Es importante destacar que de la herramienta Sonar fue tomada la idea de hacer gráficas con los resultados de las revisiones y mostrar comentarios sobre deficiencias encontradas y el lugar en que se encuentran.

Una vez confeccionadas las listas de chequeo y realizado el estudio de las herramientas similares para tener idea cómo se realizan las revisiones en el mundo se prosigue a realizar un estudio de diferentes metodologías y herramientas para desarrollar el sistema que informatizará el procedimiento de RTF en la fase de Implementación.

1.4 Metodologías de desarrollo de software

Las metodologías de desarrollo de *software* definen claramente un marco de trabajo por el que se guiará el equipo de desarrollo durante la ejecución de un proyecto. La metodología a utilizar se define como parte del trabajo que se realiza de ingeniería de *software* y constituye uno de los pasos más importantes porque condiciona todo el trabajo posterior (24).

Existen dos grandes grupos de metodologías de desarrollo de *software*, las metodologías tradicionales o robustas y ágiles. En la tabla 2 se muestra un análisis comparativo entre estos dos grupos (24).

Tabla 2. Comparación entre grupos de metodologías

Metodologías tradicionales	Metodologías ágiles
Rigidez ante los cambios, de manera lenta o moderada.	Flexibilidad ante los cambios del proyecto de forma moderada a rápida.
Los clientes interactúan con el equipo de desarrollo mediante reuniones.	Los clientes forman parte del equipo de desarrollo.
Grupos de desarrollo de gran tamaño y distribuidos en diferentes sitios.	Grupos pequeños, alrededor de diez personas, en un mismo lugar.
Dependencia de la arquitectura de *software* mediante modelos.	Menor dependencia de la arquitectura de *software.*
Poca retroalimentación lo que extiende el tiempo de entrega.	Continua retroalimentación acortando el tiempo de entrega.
Procesos controlados por políticas y normas.	Procesos menos controlados, pocas políticas y normas.
Seguimiento estricto del plan inicial de	Capacidad de respuesta ante los cambios.

desarrollo.	

La selección de la metodología a utilizar se define por la magnitud del proyecto que se desarrollará o sea un proyecto de gran envergadura por su alcance, impacto, personas que participan en su producción y presupuesto, implica la utilización de una metodología tradicional o robusta.

Por la envergadura del proyecto a desarrollar se elimina la posibilidad de utilizar una metodología robusta y se decide hacer uso de una ágil. A continuación se realiza un análisis de las metodologías ágiles más conocidas y utilizadas en la actualidad y se selecciona la que guiará el proceso de desarrollo.

1.4.1 Agile Unified Process

El Proceso Unificado Ágil en español (AUP sus siglas en inglés), de Scott Ambler[5], es una versión simplificada del *Rational Unified Process* (RUP). Describe de forma simple y fácil de entender la manera de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. AUP aplica técnicas ágiles incluyendo TDD[6].
Se centra especialmente en la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas que identifican los riesgos desde etapas iniciales del proyecto. En este sentido es relevante el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave, además se determinan los riesgos técnicos (25).

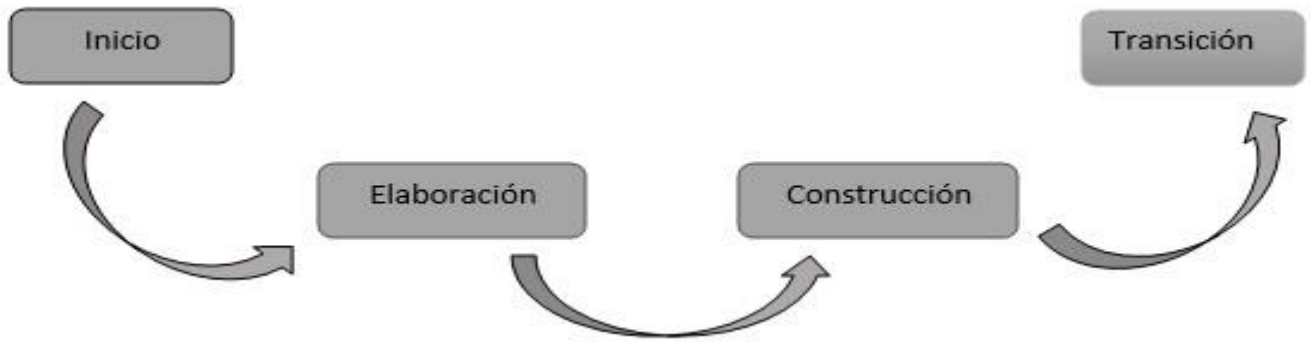


Figura1. Fases de la metodología AUP

[5] Ingeniero de Software canadiense graduado de la Universidad de Toronto.
[6] (*Test Driven Development*) en español Desarrollo Guiado por Pruebas.

Fases de AUP (26)

- Inicio: el objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.
- Elaboración: el objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
- Construcción: durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
- Transición: el sistema se lleva a los entornos de pre-producción donde se somete a pruebas de validación y aceptación, finalmente se despliega en los sistemas de producción.

Esta metodología se comenzará a utilizar en la UCI para simplificar el trabajo en los centros productivos existentes. El uso mancomunado de la misma reduce la cantidad de metodologías que hasta ahora se utilizan, esto permite una mejor comunicación entre los desarrolladores (26).

1.4.2 Xtreme Programming

Xtreme Programming (XP) o Programación Extrema en español, es una metodología de las más usadas en la actualidad por los resultados obtenidos a nivel global. Fue formulada por Kent Beck[7] quien escribió el primer libro sobre la materia llamado *Extreme Programming Explained: Embrace Change* (27).

Su característica fundamental consiste en el hecho de potenciar las relaciones entre los miembros del equipo de desarrollo, la programación en parejas es una muestra de ello. Debe existir una comunicación excelente entre cada uno de los miembros, aprovechando de esta manera el trabajo en equipo al máximo. Se caracteriza además por una simplicidad inteligente en las soluciones implementadas para satisfacer los requisitos del cliente, de forma que el trabajo desarrollado por su sencillez no constituya un gran escollo para los desarrolladores, algo simple y a la vez eficaz en la solución de un determinado requisito (28).

Se cataloga también, como una metodología donde los desarrolladores están preparados para los constantes cambios en los requisitos. Implementar soluciones simples implica cambios de menor envergadura y complejidad, válidos para un equipo de desarrollo pequeño y con alta comunicación.

[7] Ingeniero de Software estadounidense, fundador de la metodología de desarrollo de software ágil XP y Desarrollo Guiado por Pruebas (TDD por sus siglas en inglés), posee un máster en Ciencias de la Computación.

Genera pocos artefactos lo cual agiliza la ejecución del proyecto, contiene pocos roles y no se hace un énfasis notable en la arquitectura de *software* (28).

Para comprender en profundidad esta metodología es necesario entender los siguientes tres aspectos fundamentales (27):

- Historias de usuario (HU).
- Roles.
- Proceso y práctica.

Historias de usuario

Es la técnica utilizada para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente los requisitos y características que el sistema debe poseer. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada una es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (27):

Roles

Los roles de acuerdo con la propuesta original de Beck son:

- Programador.
- Cliente.
- Encargado de pruebas.
- Encargado de seguimiento.
- Entrenador (*Coach*).
- Consultor.
- Gestor (*Big boss*).

Proceso

El ciclo de desarrollo de XP consiste en la siguientes cinco etapas:

1. El cliente define el valor del negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo a sus prioridades y a las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

Es válido señalar que durante este proceso cíclico de pasos el programador y el cliente van adquiriendo experiencia mientras se desarrolla, a la vez que se establece una estrecha comunicación entre ambos (27).

Prácticas definidas para XP

Para el buen desempeño de XP es necesario tener en cuenta doce prácticas descritas a continuación.

1. El juego de la planificación.
2. Entregas pequeñas.
3. Metáfora.
4. Diseño simple.
5. Pruebas.
6. Refactorización.
7. Programación en parejas.
8. Propiedad colectiva del código.
9. Integración continua.
10. Cuarenta horas por semana.
11. Cliente in-situ.

1.4.3 SCRUM

Es usada con mucho éxito en proyectos en los que cambia con facilidad los requisitos del sistema. Son muchos los autores que han escrito sobre esta metodología pero sus fundadores fueron Ken Schwaber[8], Jeff Sutherland[9] y Mike Beedle[10].

SCRUM es una metodología de desarrollo simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto (29).

Son tres los elementos que componen el desarrollo de SCRUM (29).

1. Reuniones.
2. Elementos.

[8] Desarrollador de *software* estadounidense, uno de los líderes del movimiento de desarrollo ágil de *software*.
[9] Estadounidense graduado de la Universidad de Stanford, líder dentro del movimiento de desarrollo ágil de *software*.
[10] Es uno de los autores del manifiesto ágil.

3. Roles.

Tiene dos características distintivas que la hacen particular dentro de las metodologías de desarrollo ágiles, la primera, que el proyecto se realiza mediante iteraciones, denominados *sprints* con una duración de treinta días, el resultado de cada *sprint* es un incremento de la aplicación en funcionamiento y la segunda, la realización de reuniones constantes a lo largo de la ejecución del proyecto, entre ellas se destaca una reunión diaria de quince minutos entre el equipo de desarrollo para coordinar e integrar el trabajo diario (29).

Metodología de desarrollo seleccionada

La aplicación informática a desarrollar es de pequeña envergadura por lo que se seleccionó la metodología XP para su ejecución. El equipo de trabajo es de dos personas por lo cual se integra perfectamente con los pocos roles en el proceso de desarrollo que define esta metodología. Estas dos personas en el papel de desarrolladores más el cliente, pueden abarcar todo el trabajo de los siete roles que contiene, además el tiempo de ejecución del proyecto no debe superar los tres meses.

Existe una buena comunicación entre ambos desarrolladores y el cliente, lo cual coincide con una exigencia que define XP. Además, la aplicación a desarrollar tiene altas probabilidades de cambiar sus requisitos, problema fácilmente superable con esta metodología por la excelente cohesión que tienen los miembros y la filosofía de implementación de soluciones simples que impliquen en caso de cambios de requisitos poco impacto. También, se tuvieron en cuenta las siguientes características para su selección: alto grado de adaptabilidad, simplicidad y su desarrollo iterativo e incremental, lo que proporciona pequeñas mejoras una tras otra.

1.5 Lenguajes de programación

Un lenguaje de programación es aquella estructura que con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora (30).

1.5.1 Lenguajes del lado del cliente

Son aquellos lenguajes que son asimilados directamente por el navegador y no necesitan previo tratamiento (30).

HTML

HTML (*Hypertext Markup Language*) lenguaje de marcas de hipertexto en español, fue el lenguaje de programación escogido para el trabajo con las plantillas de las páginas web de la aplicación. Su selección por encima de otros lenguajes para el trabajo en la web como HTML 4.0 y XHTML está dado por algunas de sus características entre las que se destacan el código abierto y que el código generado es más simple. Esto permite elaborar páginas ligeras que carguen rápidamente en los navegadores, capacidad de ejecutaras sin estar conectado, mayor compatibilidad con navegadores de dispositivos móviles. Además, tiene una excelente compatibilidad con la versión 3.0 de CSS (*Cascading Style Sheet*) (31).

Se va a hacer uso de HTML v5.0 por una serie de ventajas que esta versión presenta las cuales son mencionadas a continuación (32):

- Es nativo, y por tanto independiente de *plugins* de terceros, es decir, no pertenece a nadie, es código abierto.
- Es más semántico, con etiquetas que permiten clasificar y ordenar en distintos niveles y estructuras el contenido. Además, incorpora metadatos de manera más formal, favoreciendo la accesibilidad.
- El código es más simple lo que permite hacer páginas más ligeras que se cargan más rápidamente favoreciendo la usabilidad y la indexación en buscadores.
- Ofrece una compatibilidad mayor con los navegadores de dispositivos móviles.
- Incluye la etiqueta de dibujo *canvas*, que ofrece más efectos visuales.
- Ofrece soporte a *codecs* específicos.
- Posibilita la inserción de vídeos y audio de forma directa.
- Permite la localización geográfica del usuario. Algo muy útil para la mercadotecnia móvil.
- Tiene la capacidad de ejecutar páginas sin estar conectado.
- Incorpora nuevas capacidades Javascript que aumentan la capacidad de almacenamiento. Frente a las *cookies* que dejaban almacenar algunos *kilobytes*, ahora se puede conseguir el almacenamiento de entre 5 y 10 megas, dependiendo de la plataforma. Además, se permiten múltiples Javascript corriendo en paralelo en una misma página.
- Dispone de nuevas capacidades CSS 3 como posibilidad de usar cualquier fuente o tipografía en HTML, columnas de texto, opacidad, transparencia, canales *alpha*, contraste, saturación, brillo, animaciones de transición y transformación, bordes redondeados, gradientes y sombras.
- Permite realizar diseños adaptables a distintos dispositivos (*web*, *tablets*, móviles).

CSS

Fue el lenguaje escogido para el trabajo con la presentación de la página web de la aplicación a desarrollar. Su elección estuvo dada por varias de sus ventajas al utilizarlo en su versión 3, entre las que se destacan, un mayor control de la presentación del sitio al poder tender todo el código reunido en uno, esto facilita su modificación. Además, permite mayor legibilidad del código HTML al tener el CSS en otro lugar y pueden mostrarse distintas hojas de estilo según el dispositivo que estemos utilizando (33).

JavaScript

Es un lenguaje de *scripting* multiplataforma, orientado a objetos pequeño y liviano. Dentro de un ambiente de *host*, JavaScript puede conectarse a los objetos de su ambiente y proporcionar control programático sobre ellos. JavaScript contiene una biblioteca estándar de objetos, tales como *Array*, *Date* y *Math*, además de un conjunto central de elementos del lenguaje, tales como operadores, estructuras de control y sentencias. El núcleo de JavaScript puede extenderse para varios propósitos, complementándolo con objetos adicionales, por ejemplo (34):

- *Client-side* JavaScript extiende el núcleo del lenguaje proporcionando objetos para controlar un navegador y su modelo de objetos (o DOM, por las iniciales de *Document Object Model*). Por ejemplo, las extensiones del lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, tales como *clicks* del ratón, ingreso de datos al formulario y navegación de páginas.
- *Server-side* JavaScript extiende el núcleo del lenguaje proporcionando objetos relevantes a la ejecución de JavaScript en un servidor. Por ejemplo, las extensiones del lado del servidor permiten que una aplicación se comunique con una base de datos, proporcionar continuidad de la información de una invocación de la aplicación a otra, o efectuar manipulación de archivos en un servidor.

Se usa este lenguaje porque tiene ventajas como (34):

- Es un lenguaje muy sencillo.
- Tiene gran documentación en la web y es totalmente gratuito.
- Es liviano y usarlo podrá crear páginas web dinámicas, menús desplegables, efectos visuales sencillos, manipular datos y crear aplicaciones web, utilizando poca memoria y manteniendo un tiempo de descarga rápido para tu página web.

XML

Su nombre proviene de *Extensible Markup Language* (Lenguaje de Marcas Extensible). Se trata de un metalenguaje (un lenguaje que se utiliza para decir algo acerca de otro) extensible de etiquetas que fue desarrollado por el *Word Wide Web Consortium* (W3C), una sociedad mercantil internacional que elabora recomendaciones para la *World Wide Web* (35).

El XML es una adaptación del SGML (*Standard Generalized Markup Language*), un lenguaje que permite la organización y el etiquetado de documentos. Esto quiere decir que el XML no es un lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo a las necesidades. El XHTML, el MathML y el SVG son algunos de los lenguajes que el XML tiene la capacidad de definir (35).

Este lenguaje es utilizado porque tiene muchas ventajas de uso, algunas de estas se mencionan a continuación (36):

- Fácilmente procesable tanto por humanos como por *software*.
- Separa radicalmente la información o el contenido de su presentación o formato.
- Diseñado para ser utilizado en cualquier lenguaje o alfabeto.
- Su análisis sintáctico es fácil debido a las estrictas reglas que rigen la composición de un documento.
- Estructura jerárquica.
- El número de marcas es ilimitado.

UML

Lenguaje Unificado de Modelado (UML por sus siglas en inglés), es un popular lenguaje de modelado de sistemas de *software*. Se trata de un lenguaje gráfico para construir, documentar, visualizar y especificar un sistema informático. Posee la riqueza suficiente como para crear un modelo del sistema, pudiendo modelar los procesos de negocios, funciones, esquemas de bases de datos y expresiones de lenguajes de programación. Para ello utiliza varios tipos diferentes de diagramas, por ejemplo, en UML 2.0 hay trece tipos de diagramas. Este lenguaje de modelado es seleccionado porque este proporciona ventajas tales como la comunicación sencilla y rápida entre desarrolladores y clientes. Además, no se necesitan conocimientos profundos de ingeniería de *software* para que los clientes comprendan lo que los desarrolladores muestran, de modo que rápidamente pueden expresar su conformidad con el producto o las nuevas mejoras que desean ver introducidas (37).

1.5.2 Lenguajes del lado del servidor

Son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él (28).

PHP

Es un lenguaje de código abierto muy popular, especialmente adecuado para el desarrollo web que puede ser incrustado en HTML, fue creado por Rasmus Lerdorf[11].

Se selecciona como el lenguaje de programación para trabajar del lado del servidor en su versión 5.4.32 por toda una serie de ventajas que mencionamos a continuación.

Algunas características representativas de PHP (38):

- Es un lenguaje multiplataforma.
- Fácil de aprender y de código abierto (*open source*).
- Orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- Tiene manejo de excepciones desde PHP v5.

1.6 Framework de desarrollo

Se puede definir *framework* como un marco de trabajo para el desarrollo de aplicaciones informáticas que agiliza y mantiene armonía en el proceso de su construcción. Estos no necesariamente están ligados a un lenguaje de programación en concreto y aunque sea así en muchas ocasiones, nada impide definir el mismo *framework* para lenguajes diferentes. Cuanto más detallado es, más necesidad tendrá de ajustarse a un lenguaje concreto. También, es posible que defina una estructura para una aplicación completa o bien sólo se centre en un aspecto de ella.

[11] Programador informático danés, graduado de la Universidad de Waterloo creador de la primera versión del lenguaje de programación PHP.

Entre las ventajas más evidentes de la utilización de estas herramientas se pueden mencionar (39):

- El programador no necesita plantearse una estructura global de la aplicación, sino que él le proporciona un esqueleto que hay que rellenar.
- Facilita la colaboración.
- Proporciona un mejor entendimiento entre los desarrolladores pues el código se genera siguiendo estandarizaciones definidas previamente.
- Es más fácil encontrar herramientas (utilidades, bibliotecas) adaptadas al concreto para facilitar el desarrollo.

1.6.1 Symfony

Symfony es un *framework* de PHP basado en la arquitectura MVC (Modelo-Vista-Controlador), su primera versión surgió en el año 2005 como un proyecto de *software* libre, el cual se ha convertido en uno de *framework* más populares de PHP que existe en la actualidad, su fundador y mayor impulsor dedica la mayor parte de su tiempo a corregir los problemas que puedan existir y que son identificado por la comunidad de la cual se rodea este *framework* (40).

A continuación de mencionan un conjunto de características de este *framework* las cuales fundamentan el por qué se seleccionó el mismo para desarrollar la herramienta:

1. Symfony integra dos productos muy potentes ya existentes en el mercado, que podríamos haber instalado de forma individual en nuestra aplicación (41):

- Twig: un motor de plantillas rápido, seguro y flexible que resuelve a la perfección la separación de la representación de las vistas del patrón MVC.
- Doctrine: un ORM (mapeador de objetos relacional) que facilitará el trabajo a la hora de interactuar con la capa de persistencia en base de datos.

2. Symfony organiza las funcionalidades en paquetes o Bundles (en su propia terminología). En caso que se necesite una funcionalidad que no venga por defecto con el *framework* lo más probable es que alguien haya tenido antes la misma necesidad, así que antes de implementarla se debe consultar si existe algún Bundle que se pueda instalar en la aplicación que la resuelva.
3. Symfony sigue un conjunto de buenas prácticas determinadas por la comunidad para el desarrollo de *software*, como son:
    - Programación orientada a objetos.

- Modelo Vista Controlador.
- Inyección de dependencias.
- Mapeador de objetos relacional.
- Internacionalización.
- Mecanismo de caché.
- Autentificación y autorización.
- Sistemas de seguridad para evitar ataques por fuerza bruta, CSRF e inyecciones SQL.
- Integración con pruebas unitarias y de comportamiento.

4. Al ser tan popular, los entornos integrado de desarrollo (IDE) más utilizados se han preparado para mejorar la integración y la experiencia de desarrollo, reconociendo *code completion, coding style*, atajos para acceder a la definición de métodos, clases y ficheros de configuración, como es el caso de los *plugins* para Netbeans, Eclipse, Sublime, y PhpStorm. En el entorno de desarrollo, se incluye automáticamente en el pie de cada página una barra de depuración y de perfiles de gran utilidad, pudiendo acceder desde ahí a información relevante para el desarrollador. Mediante la consola de comandos se puede ejecutar el proceso *batch*, tanto desarrollados por nosotros, como funcionalidades propias de Symfony.

Teniendo en cuenta todos los elementos antes enunciados se puede afirmar que Symfony2 tiene todo lo necesario para poder abordar el desarrollo de una aplicación seria y de envergadura con garantías.

1.7 Herramientas CASE

Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos 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 costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores (42).

1.7.1 Visual Paradigm

Como herramienta de modelado de sistemas es seleccionada Visual Paradigm en su versión 8.0 porque es una de las más destacadas de las llamadas herramientas CASE. Es utilizada para modelado

UML. La herramienta está diseñada para una amplia gama de usuarios, incluidos los ingenieros de *software*, analistas de sistemas, analistas de negocios y arquitectos de sistemas o para interesados en la construcción de sistemas de *software* fiable a gran escala con un enfoque orientado a objetos. Además, Visual Paradigm-UML soporta los últimos estándares de la notación UML (43).

1.8 Entorno de Desarrollo Integrado

Un IDE es un programa informático compuesto por un conjunto de herramientas de programación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Estos pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación (44).

Entre los IDE más utilizados a nivel mundial se encuentran NetBeans, VisualStudio, Eclipse, Sublime Text, PhpStorm y ZendStudio. NetBeans y PhpStorm son los más populares para el trabajo con Symfony. A continuación se hace un análisis con las principales características, ventajas y desventajas de ambas herramientas de desarrollo.

1.8.1 PhpStorm

Es un IDE desarrollado por JetBrains. Es optimizado para el desarrollo de aplicaciones web pues es cómodo el trabajo con lenguajes como HTML 5, JavaScript, CSS; otra de sus ventajas es que soporta la mayoría de los *framework* de desarrollo PHP (Symfony, Zend, CodeIgniter). Tiene autocompletado de código, algo muy agradecido por los desarrolladores. Entre sus desventajas se puede citar el hecho de que en comparación con NetBeans consume más recursos de *hardware,* además de no entrar en la categoría de *software* libre para el desarrollo de aplicaciones (45).

1.8.2 NetBeans

NetBeans es un entorno integrado de desarrollo libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo, es un producto gratuito sin restricciones de uso.

Es un proyecto de código abierto de gran éxito, con una gran base de usuarios y una comunidad en constante crecimiento. La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de *software* llamados módulos, que pueden ser extendidas agregándole otros. Este IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, *web*, EJB y aplicaciones móviles) (46).

IDE seleccionado

Fue el IDE seleccionado en su versión 8.0 para el desarrollo de la aplicación por ser completamente de código abierto, brinda la posibilidad de agregar nuevo módulos para aumentar sus funcionalidades y cuenta con abundante documentación de fácil acceso. Tiene excelente compatibilidad con Symfony v.2 en adelante y por ende con todos los lenguajes que se utilizan para el desarrollo de una herramienta informática de tipo web. Para agilizar el trabajo de los desarrolladores cuenta con autocompletado de código PHP y HTML y una consola para el trabajo en Symfony que autocompleta todos los comandos que se utilizan en el *framework* y consume pocos recursos comparado con otros IDE de programación (47).

1.9 Sistema gestor de base de datos

Los sistemas gestores de base de datos (SGBD) son un tipo de *software* muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición y manipulación de datos y uno de consultas (48).

1.9.1 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[12] y con su código fuente disponible libremente. Utiliza un modelo cliente-servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (49).

En la UCI el uso de este SGBD es muy extendido debido a que va con las políticas de la universidad y del país a potenciar el uso de tecnologías de código abierto, con el objetivo de alcanzar la soberanía tecnológica tan necesaria en un mundo donde la informática y las telecomunicaciones se presentan como un nuevo escenario de batalla entre las naciones, empresas y entidades de todo tipo. Su selección como SGBD en su versión 9.2.4 estuvo determinada fundamentalmente por el hecho de soportar grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. Cuenta con una confiabilidad y estabilidad legendarias, alcanzadas por su buen desempeño en el manejo de grandes volúmenes de datos (49).

[12] (*Berkeley Software Distribution*) Es una licencia de *software* libre permisiva, es menos restrictiva en comparación con otras por el hecho de permitir el uso de código fuente en *software* no libre.

1.10 Sistema de administración para PostgreSQL

1.10.1 Pgadmin

Es una aplicación de diseño y manejo de bases de datos para su uso con PostgreSQL. La aplicación se puede utilizar para manejar PostgreSQL 7.3 y superiores y funciona sobre casi todas las plataformas. Este *software* fue diseñado para responder a las necesidades de todos los usuarios, desde la escritura de simples consultas SQL a la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de PostgreSQL y facilita la administración. La aplicación también incluye un editor de la sintaxis SQL y un editor de código del lado del servidor. La conexión del servidor se puede realizar mediante TCP/IP o *Unix Domain Sockets* y puede ser cifrado mediante SSL por seguridad. No se requieren controladores adicionales para comunicarse con la base de datos del servidor. Una característica interesante de PgAdmin es que, cada vez que se realiza alguna modificación en un objeto, escribe la(s) sentencia(s) SQL correspondiente(s), lo que hace que, además de una herramienta muy útil, sea a la vez didáctica. También, incorpora funcionalidades para realizar consultas, examinar su ejecución (como el comando *explain*) y trabajar con los datos. Por estas ventajas y facilidades de uso fue el sistema de administración de bases de datos seleccionado en su versión 3 (50) (51).

1.11 Servidor de base de datos

Un servidor de base de datos es un programa que provee servicios de base de datos a otros programas u otras computadoras, como es definido por el modelo cliente - servidor (52).

1.11.1 Apache

El servidor Apache es un servidor web de código abierto, para plataformas *Unix*, Microsoft Windows y Macintosh que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Se desarrolla dentro del proyecto *HTTP Server* de la *Apache Software Foundation*.

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración. Es usado principalmente para enviar páginas web estáticas y dinámicas en la *World Wide Web*. Muchas aplicaciones web están diseñadas asumiendo como ambiente de implantación a Apache o que utilizarán características propias de este servidor web.

Es usado para otras tareas donde el contenido necesita ser puesto a disposición en una forma segura y confiable. Un ejemplo es al momento de compartir archivos desde una computadora personal hacia Internet. Un usuario que tiene Apache instalado en su escritorio puede colocar arbitrariamente archivos en la raíz de sus documentos, desde donde pueden ser compartidos. Los programadores de aplicaciones web a veces utilizan una versión local de Apache con el fin de obtener una vista previa y probar código mientras éste es desarrollado (53).

Entre las ventajas que propiciaron que este fuera el servidor seleccionado se encuentran (53):

- Modular
- Código abierto
- Multi-plataforma
- Extensible
- Popular

Conclusiones del capítulo

Luego del estudio y análisis de los conceptos fundamentales de la investigación y de aplicaciones similares, las metodologías de desarrollo, las tecnologías y herramientas, se concluye que:

- Es importante asegurar la calidad de *software* a través de la realización de revisiones para lograr un producto que se ajuste a las necesidades crecientes de los clientes del mundo contemporáneo.
- Existen muchas herramientas para la revisión de código pero estas no cuentan con las posibilidades de configuración necesarias para adaptarse a los estándares de calidad definidas por el centro FORTES en el área de Implementación del desarrollo de *software*.
- Entre las metodologías de desarrollo de *software* analizadas se seleccionó para guiar el proceso de desarrollo XP por sus características, fundamentalmente porque se ajusta con el tipo de producto que se va a desarrollar y la composición del equipo de desarrollo.

Capítulo 2. Presentación de la propuesta de solución.

Introducción

En este capítulo se elabora la propuesta de solución guiado por las etapas de la metodología XP. Se exponen los principales artefactos generados en cada una de las etapas con su descripción tales como: historias de usuario, tarjetas CRC (Clase, Responsabilidad y Colaboración) y tareas de ingeniería, necesarios para el desarrollo de la implementación del producto. También, se expone y describe el modelo de datos generado, el patrón arquitectónico y los de diseños utilizados.

2.1 Propuesta de solución

El módulo a implementar tendrá la responsabilidad de verificar la calidad durante la fase de Implementación. Se enfoca en las necesidades del centro productivo FORTES de la UCI, o sea que se va a regir por los parámetros de calidad que define este centro, dígase los estándares de codificación para cada uno de los lenguajes que se utilizan en el desarrollo de sus aplicaciones. Se asegurará así un código fuente de calidad y en la revisión de la adecuada elaboración de la documentación generada en la fase. Además, permitirá una gestión inexistente en la actualidad del resultado de las revisiones. El resultado del chequeo de la calidad en esta fase deberá emitir un informe con todas las no conformidades detectadas, llamando a su erradicación para evitar errores futuros.

Está concebido para llevar un registro preciso de todas las RTF que se realicen sobre un proyecto. Deberá contar con dos roles, uno de administración y otro un usuario sin permisos de administración. El usuario de tipo administrador tendrá la capacidad de gestionar nueve entidades: usuario, tipo de proyecto y tipos documentos, lenguaje de programación, tipo de deficiencia de los documentos y archivos, complejidad de deficiencia, lista de chequeo de los documentos y los lenguajes. Esto propiciará flexibilidad al sistema para adaptarse a las nuevas condiciones del entorno.

El usuario deberá ser un revisor con experiencia en implementación de diferentes lenguajes de programación. Todas las funcionalidades ejecutables por este último son exclusivamente las relacionadas con la ejecución de la RTF. A continuación se exponen los pasos para ejecutar una RTF en la aplicación.

El usuario autenticado registrará un proyecto sobre el cual se ejecutará la revisión en caso de no estar registrado. Posteriormente insertará llenando los campos del formulario de una nueva RTF sobre el proyecto. Luego registrará todos los componentes a revisar. Existen dos tipos de componentes, documentos y archivos. Los documentos inicialmente pueden ser de tipo Manual de usuario y Guía de instalación y configuración, ambos generados durante la fase de Implementación. Los archivos que se revisarán se pertenecen al código fuente de la aplicación.

Una vez registrados los componentes, se procederá a revisar cada uno de ellos. Cada tipo de documento tendrá asociado una lista de chequeo para comprobar su correcta elaboración. En el caso de los archivos en dependencia del lenguaje de programación será la lista de chequeo que compruebe el estricto cumplimiento del estándar de codificación definido. La aplicación reconocerá de manera automática en caso de que el lenguaje del código fuente sea PHP las deficiencias más significativas encontradas en el código fuente que se revisa, humanizando de esta manera el trabajo de identificarlas por parte del revisor.

Ante la detección de una deficiencia en alguno de los componentes, el usuario podrá registrarla, llenando una serie de parámetros que varían en dependencia del tipo de componente. Una vez concluida la revisión de cada uno de los componentes se dará por concluida la RTF y la aplicación mostrará una tabla resumen con los resultados de la misma.

La aplicación desarrollada también deberá contar con un módulo estadístico que se nutre de los resultados de las RTF aplicadas a los proyectos. Este presenta, a través de gráficas, datos que sirven de alerta al revisor y para que conozca donde son más frecuentes las deficiencias en los archivos y documentos, en dependencia de su lenguaje de programación y tipo de documento.

En la figura 2 se muestra el mapa de navegación de la herramienta:

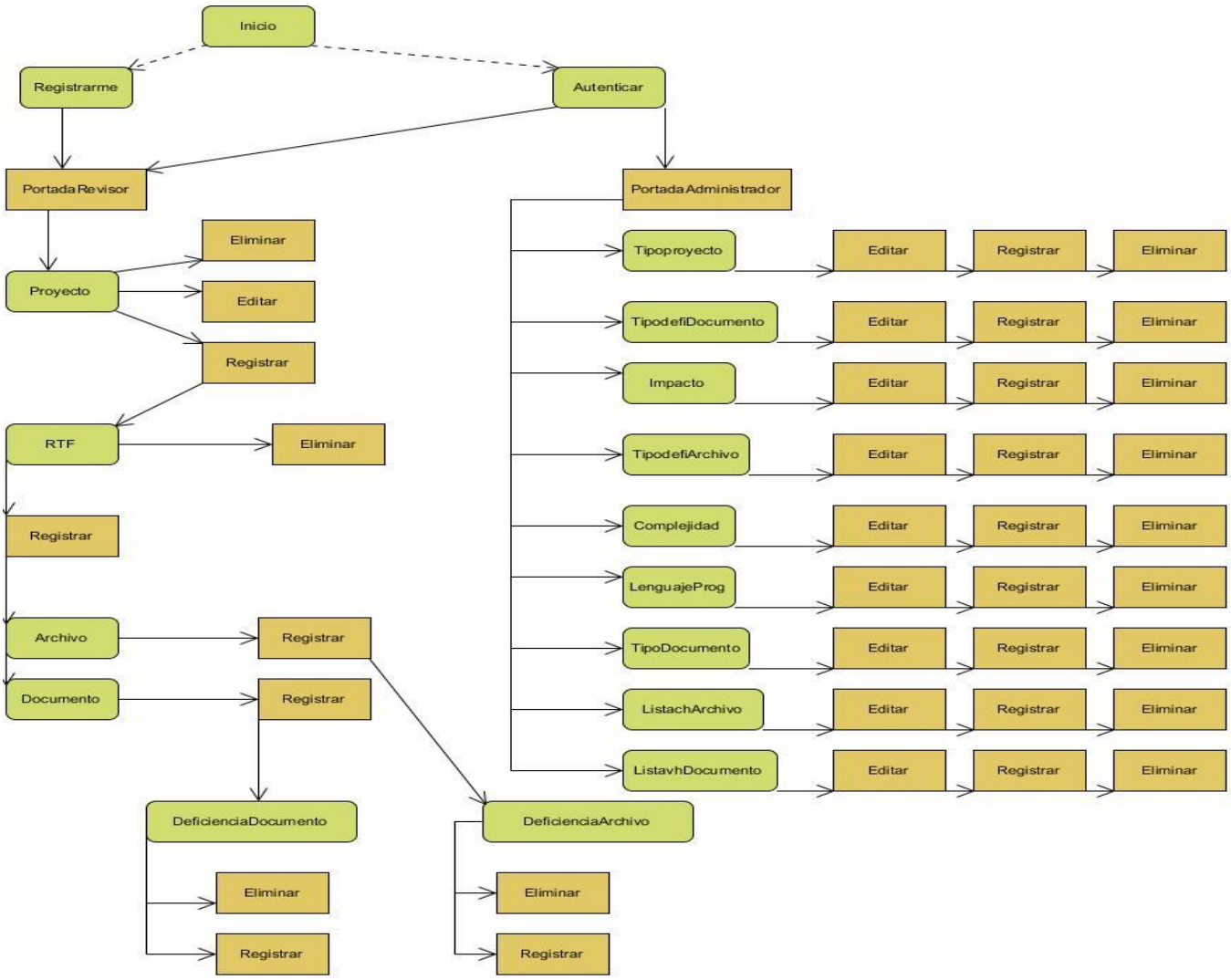


Figura 2. Mapa de navegación

2.2 Exploración

Esta constituye la primera fase cuando se desarrolla bajo la metodología XP. En la misma el cliente define a grandes rasgos cuales son las funcionalidades de la aplicación de mayor interés para priorizar su ejecución por parte del equipo de desarrollo. Estas funcionalidades se representan a través de las historias de usuario de una forma sencilla que facilite su comprensión. La exploración debe durar unas pocas semanas en dependencia de la familiarización que tengan los desarrolladores con las tecnologías, prácticas y herramientas a utilizar.

2.2.1 Historias de usuario

Son pequeñas pero concisas formas de describir las funcionalidades del sistema. Las mismas son la base para ejecutar las pruebas de aceptación. Están compuestas fundamentalmente por: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica, cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas, lista de seguimiento con fecha, estados de cosas por terminar y comentarios. A continuación se muestran los parámetros de estas (27):

Parámetros de clasificación de las historias de usuario

Prioridad en el negocio: este parámetro es definido por el cliente y puede obtener uno de los siguientes tres valores:

- Alta: se le otorga a las funcionalidades que el cliente define como de alta prioridad, fundamentales para el producto.
- Media: se le otorga a las funcionalidades que resultan para el usuario a tener en cuenta siempre que no afecten el desarrollo central del sistema.
- Baja: se le otorga a las funcionalidades de menor peso dentro del sistema, aquellas que no tienen una importancia notable dentro del producto.

Riesgo en el desarrollo: este parámetro es definido por los desarrolladores y puede obtener uno de los siguientes valores:

- Alto: se otorga cuando la implementación de esa funcionalidad puede conllevar errores que hagan inoperable el código de la aplicación.
- Medio: se otorga cuando pueden aparecer errores en la implementación de la funcionalidad que impliquen el retraso en la entrega del producto.
- Bajo: se otorga cuando se pueden producir errores fácilmente solucionables que no tengan una gran repercusión negativa en la ejecución de la aplicación.

Puntos estimados: estos los definen los desarrolladores a partir de un análisis de la complejidad de la funcionalidad y su valor está dado por la cantidad de semanas necesarias para su ejecución.

Puntos reales: estos constituyen la duración real en semanas de la implementación de la historia de usuario.

A continuación se presentan algunos ejemplos de estas, las demás se encuentran en los anexos (ver anexo 1).

Tabla 3. Historia de usuario #1

Historia de Usuario	
Número: 1	**Nombre:** Gestionar proyecto
Usuario: Usuario normal	**Iteración:** 1
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alt**o**	**Puntos reales:** 1/3
Descripción: La aplicación debe permitir registrar, editar y eliminar un proyecto sobre el cual se realizarán las revisiones.	

De los requisitos funcionales definidos, once de ellos son de gestión de entidades que solamente pueden ser realizados por el administrador del sistema. Estas entidades no son más que los tipos de valor que pueden obtener los parámetros de otras entidades de la aplicación.

La HU presentada a continuación posibilita que si en un momento surgiera una nueva categoría de proyecto en el centro FORTES, la aplicación mediante la gestión del administrador del sistema pueda añadir esta nueva categoría de manera automática en los formularios de registro de proyectos.

Tabla 4. Historia de usuario #9

Historia de Usuario	
Número: 9	**Nombre:** Gestionar tipo de proyecto
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1**/**2
Descripción: La aplicación debe permitir registrar y editar los tipos de proyectos que contiene para clasificar el sistema.	

Así de la misma forma las siguientes dos historias de usuario permiten algo similar pero con los parámetros tipo de documento y lenguaje de programación, necesarios para insertar un documento y un archivo al sistema respectivamente.

Tabla 5. Historia de usuario #10

Historia de Usuario	
Número: 10	**Nombre:** Gestionar tipo de documento
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y editar los tipos de documentos que contiene para clasificar el sistema.	

Tabla 6. Historia de usuario #11

Historia de Usuario	
Número: 11	**Nombre:** Gestionar lenguaje de programación
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y editar los lenguajes de programación que contiene para clasificar el sistema.	

Requisitos del sistema de tipo estadísticos

Se identificaron un grupo de requisitos del sistema de tipo estadísticos presentados en las siguientes historias de usuario. Su presencia en el sistema responde a la necesidad de mostrar los resultados históricos de las RTF. Los gráficos permitirán al revisor tener una fuente clara de identificación de los puntos más críticos a la hora de ejecutar la RTF, ganando en experiencia con la información acumulada. La siguiente historia de usuario es un ejemplo:

Tabla 7. Historia de usuario #18

Historia de Usuario	
Número: 18	**Nombre:** Graficar cantidad promedio de deficiencias por componente por tipo de componente
Usuario: Usuario normal	**Iteración:** 3

Prioridad en el negocio: Media	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe permitir mostrar una gráfica con la cantidad promedio de deficiencias identificadas por cada uno de los componentes revisados por tipo dígase documentos y archivos.	

La siguiente gráfica es una muestra del resultado de la HU anteriormente mencionada. En la misma se muestra la información de la cantidad promedio de deficiencias identificadas por cada uno de los componentes revisados teniendo en cuenta su tipo (documento o archivo).

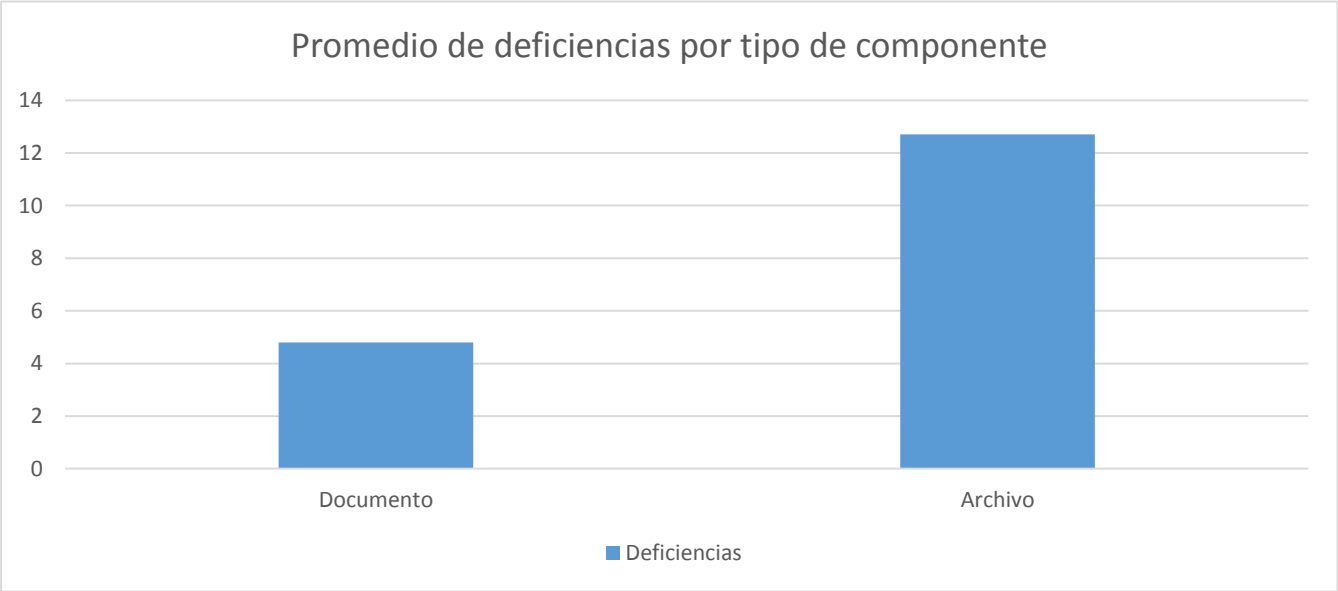


Figura 3. Gráfico de tipos de componentes

A continuación se expone otro requisito estadístico del sistema.

Tabla 8. Historia de usuario #19

Historia de Usuario	
Número: 19	**Nombre:** Graficar cantidad promedio de deficiencias por lenguaje por tipo de deficiencia
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Media	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe permitir mostrar una gráfica con la cantidad promedio de	

deficiencias identificadas por lenguaje por tipo de deficiencia.

Esta última gráfica es el producto de otra HU estadística del sistema, con esta fácilmente el revisor puede identificar por ejemplo que el lenguaje de programación PHP es más propenso a tener errores en las declaraciones de variables con respecto a los demás.

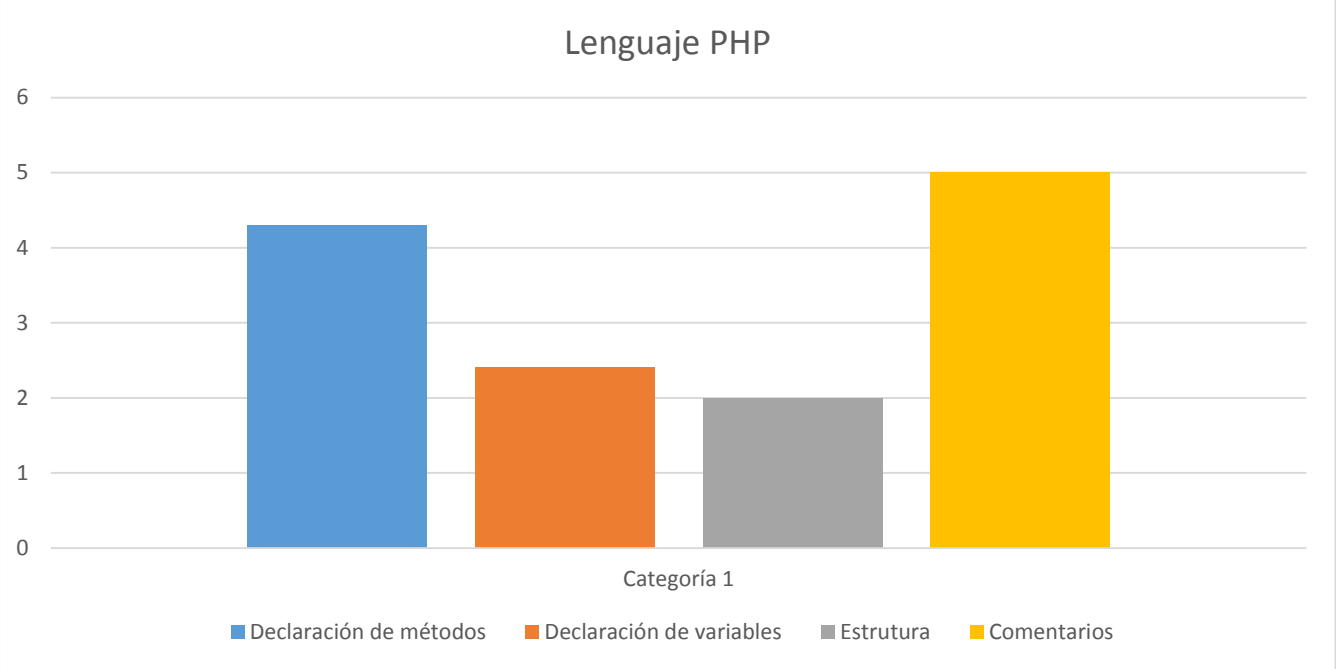


Figura 4. Gráfico de lenguajes de programación

La siguiente historia de usuario representa el requisito de sistema de mayor importancia en la herramienta. Su implementación permite que el sistema identifique las deficiencias de estandarización de código fuente en lenguaje de programación PHP más significativas de manera automática. De esta forma se obtiene una humanización del proceso de identificación de deficiencias y una disminución notable en el tiempo de realización de la RTF a la Implementación.

Tabla 9. Historia de usuario #24

Historia de Usuario	
Número: 24	**Nombre:** Mostrar posibles deficiencias identificadas en archivo de lenguaje PHP
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Alta	**Puntos estimados:** 3

Riesgo en el desarrollo: Medio	**Puntos reales:** 3
Descripción: La aplicación debe ser capaz de mostrar un listado de posibles deficiencias identificadas en el archivo con código fuente PHP a revisar.	

2.3 Planificación

Este es el momento en la metodología XP donde se define por parte del cliente el orden de prioridad de las funcionalidades del sistema. A partir de esto, los desarrolladores definen para cada historia de usuario un esfuerzo estimado fundamentado en la complejidad de la funcionalidad. También, se define el plan de iteraciones para el desarrollo del sistema. Además, se lleva muy de cerca la velocidad de desarrollo para evitar retrasos que afecten al cliente.

2.3.1 Estimación de esfuerzo

Las historias de usuario que se refieren a gestión de entidades se estima que acarreen muy poco esfuerzo por parte de los desarrolladores, pues Symfony el *framework* de desarrollo web utilizado, permite ejecutar estas funcionalidades en poco tiempo y de una manera muy sencilla. Todos los requisitos del sistema estadísticos y el de identificación de deficiencias de manera automática se estiman con un esfuerzo alto por su complejidad. A partir de esto se definió una estimación de esfuerzo para cada historia de usuario de la siguiente forma:

Tabla 10. Estimación de esfuerzo

No.	Historia de usuario	Puntos de esfuerzo
1	Gestionar proyecto	1
2	Gestionar archivo	1
3	Gestionar documento	1
4	Gestionar RTF	1
5	Gestionar deficiencia de archivo	2
6	Gestionar deficiencia de documento	2
7	Gestionar usuario	2
8	Gestionar impacto	1
9	Gestionar tipo de proyecto	1
10	Gestionar tipo de documento	1
11	Gestionar lenguaje de programación	1

12	Gestionar tipo de deficiencia de archivo	1
13	Gestionar tipo de deficiencia de documento	1
14	Gestionar complejidad	1
15	Gestionar lista de chequeo de documento	1
16	Gestionar lista de chequeo de archivo	1
17	Graficar cantidad de deficiencias por proyecto, por componente revisado	3
18	Graficar cantidad promedio de deficiencias por componente, por tipo de componente	3
19	Graficar cantidad promedio de deficiencias por lenguaje por tipo de deficiencia	3
20	Graficar cantidad promedio de deficiencias por lenguaje por componente	3
21	Generar tabla resumen de RTF	2
22	Graficar resultados de RTF	3
23	Generar Informe de RTF	3
24	Mostrar posibles deficiencias identificadas del lenguaje PHP.	3

2.3.2 Plan de iteraciones

El cliente determinó que los requisitos del sistema relacionados directamente con el proceso de RTF fueran los primeros en ejecutarse por su importancia, como segundo grupo definió los referentes a la administración del sistema y por último los requisitos de tipo estadísticos. A partir de esto el plan de iteraciones quedó de la siguiente forma abarcando un total de veintidós semanas:

Tabla 11. Plan de iteraciones

Iteración	Historia de usuario	Duración	Total
1	Gestionar proyecto	½	6 semanas
	Gestionar archivo	½	
	Gestionar documento	½	
	Gestionar RTF	¾	
	Gestionar deficiencia de archivo	½	

	Gestionar deficiencia de documento	½	
2	Gestionar usuario	¾	6 semanas
	Gestionar impacto	½	
	Gestionar tipo de proyecto	½	
	Gestionar tipo de documento	½	
	Gestionar lenguaje de programación	½	
	Gestionar tipo de deficiencia de archivo	½	
	Gestionar tipo de deficiencia de documento	½	
	Gestionar complejidad de deficiencia	½	
	Gestionar lista de chequeo de documento	½	
	Gestionar lista de chequeo de archivo	½	
3	Graficar deficiencias por proyecto por tipo de componente	1	10 semanas
	Graficar cantidad de deficiencias por tipo de componente	1	
	Graficar por cantidad de deficiencias por lenguaje por tipo de deficiencia	1	
	Graficar cantidad promedio de deficiencias por lenguaje por componente	1	
	Generar tabla resumen de RTF	1	
	Graficar resultado de RTF	1	
	Generar Informe de RTF	1	
	Mostrar posibles deficiencias identificadas del lenguaje PHP.	3	

2.4 Diseño

Cuando la fase de diseño se lleva a cabo bajo la metodología XP se caracteriza por la simpleza en los entregables generados. Se procura hacer todo lo menos complicado posible para conseguir un diseño fácilmente entendible y que sea posible de implementar que a la larga costará menos tiempo y esfuerzo desarrollar. La metodología utilizada no especifica ninguna técnica de modelado. Se utilizan indistintamente sencillos esquemas, diagramas de clases utilizando UML o tarjetas CRC (Clase, Responsabilidad y Colaboración).

2.4.1 Patrón arquitectónico

La arquitectura MVC (Modelo Vista Controlador) será la utilizada para el desarrollo del producto ya que el *framework* utilizado organiza su trabajo bajo la misma. Su uso es extendido en todo el mundo por los desarrolladores web por las grandes ventajas que ofrece, entre las que podemos mencionar:

- Clara y muy práctica separación entre los modelos, las vistas y las clases controladoras, de ahí su nombre.
- Sencillez para crear las distintas representaciones de los mismos datos.
- Facilidad para realizar pruebas unitarias de los componentes.
- Reutilización de los componentes.
- Simplicidad en el mantenimiento de los sistemas.

El patrón MVC consiste en tres tipos de objetos: el modelo que es el objeto de la aplicación, la vista que es su representación y el controlador que define el modo en que la interfaz reacciona a la entrada del usuario. De una manera poco ortodoxa podemos decir que el controlador controla el flujo de la aplicación, pide al modelo aquello que el usuario solicita y le devuelve (al usuario) una representación del modelo a través de la vista. El figura 5 ilustra la cooperación entre estos tres objetos en Symfony (54):

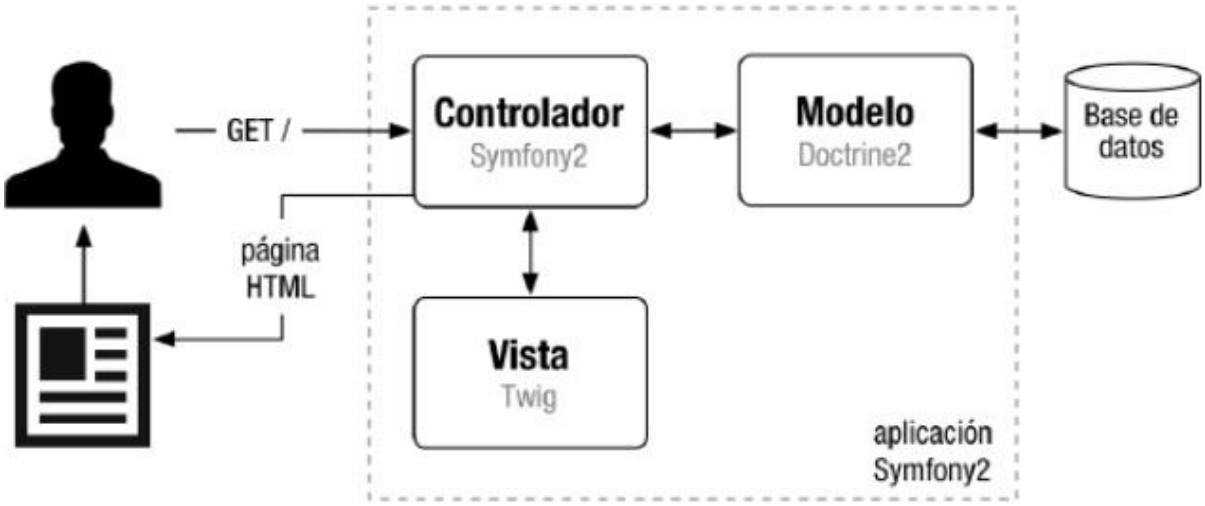


Figura 5. Patrón MVC en Symfony

2.4.2 Tarjetas CRC

Cada tarjeta CRC representa una clase en la programación orientada a objetos y define sus responsabilidades y las colaboraciones. Las responsabilidades son los métodos con los que cuenta y las colaboraciones las demás clases con las que interactúa. Estas ayudan al equipo de trabajo a definir actividades durante el diseño del sistema.

El framework de desarrollo web Symfony crea a base de comandos todos los métodos *set* y *get* de las clases entidades, ambos tipos de métodos conocidos universalmente por los programadores por su capacidad de obtención y modificación de los atributos de una entidad. Además, por cada CRUD (*Create, Replace, Update, Delete*) definido sobre una determinada entidad se crea una clase controladora. A continuación se muestran dos ejemplos de tarjetas CRC de las clases entidades RTF y Deficienciaarchivo, el resto se encuentra en los anexos (ver anexo 2):

Tabla 12. Tarjeta CRC #1

Clase: RTF	
Responsabilidades**:**	Colaboradores:
➢ Obtener fecha de inicio. ➢ Modificar fecha de inicio. ➢ Obtener fecha de terminación. ➢ Modificar fecha de terminación. ➢ Obtener el objetivo. ➢ Modificar el objetivo. ➢ Obtener el identificador del usuario que la registró. ➢ Modificar el identificador del usuario que la registró. ➢ Obtener el identificador del proyecto al cual pertenece. ➢ Modificar el identificador del proyecto al cual pertenece.	➢ Usuario ➢ Proyecto

Tabla 13. Tarjeta CRC #2

Clase: Deficienciaarchivo	
Responsabilidades**:**	Colaboradores:
➢ Obtener el asunto de la deficiencia. ➢ Modificar el asunto de la deficiencia. ➢ Obtener la fecha de registro. ➢ Modificar la fecha de registro. ➢ Obtener la descripción de la deficiencia. ➢ Modificar la descripción de la deficiencia. ➢ Obtener la ruta del archivo guardado. ➢ Modificar la ruta del archivo guardado. ➢ Obtener la línea de código. ➢ Modificar la línea de código. ➢ Obtener el identificador de la RTF. ➢ Modificar el identificador de la RTF. ➢ Obtener el identificador de la complejidad de la deficiencia. ➢ Modificar el identificador de la complejidad de la deficiencia. ➢ Obtener el identificador del tipo de deficiencia. ➢ Modificar el identificador del tipo de deficiencia. ➢ Obtener el identificador del archivo al cual pertenece la deficiencia. ➢ Modificar el identificador del archivo al cual pertenece la deficiencia.	➢ Rtf ➢ Complejidad ➢ Tipodeficienciaarchivo ➢ Archivo

2.4.3 Modelo de datos

Un modelo de datos es la representación de las relaciones que existen entre los datos en un SGBD. Este está compuesto básicamente por entidades con sus atributos y relaciones entre estas. El modelo de datos arrojado del análisis de la realización de una RTF a la Implementación en el centro FORTES contiene dieciocho tablas y es el siguiente:

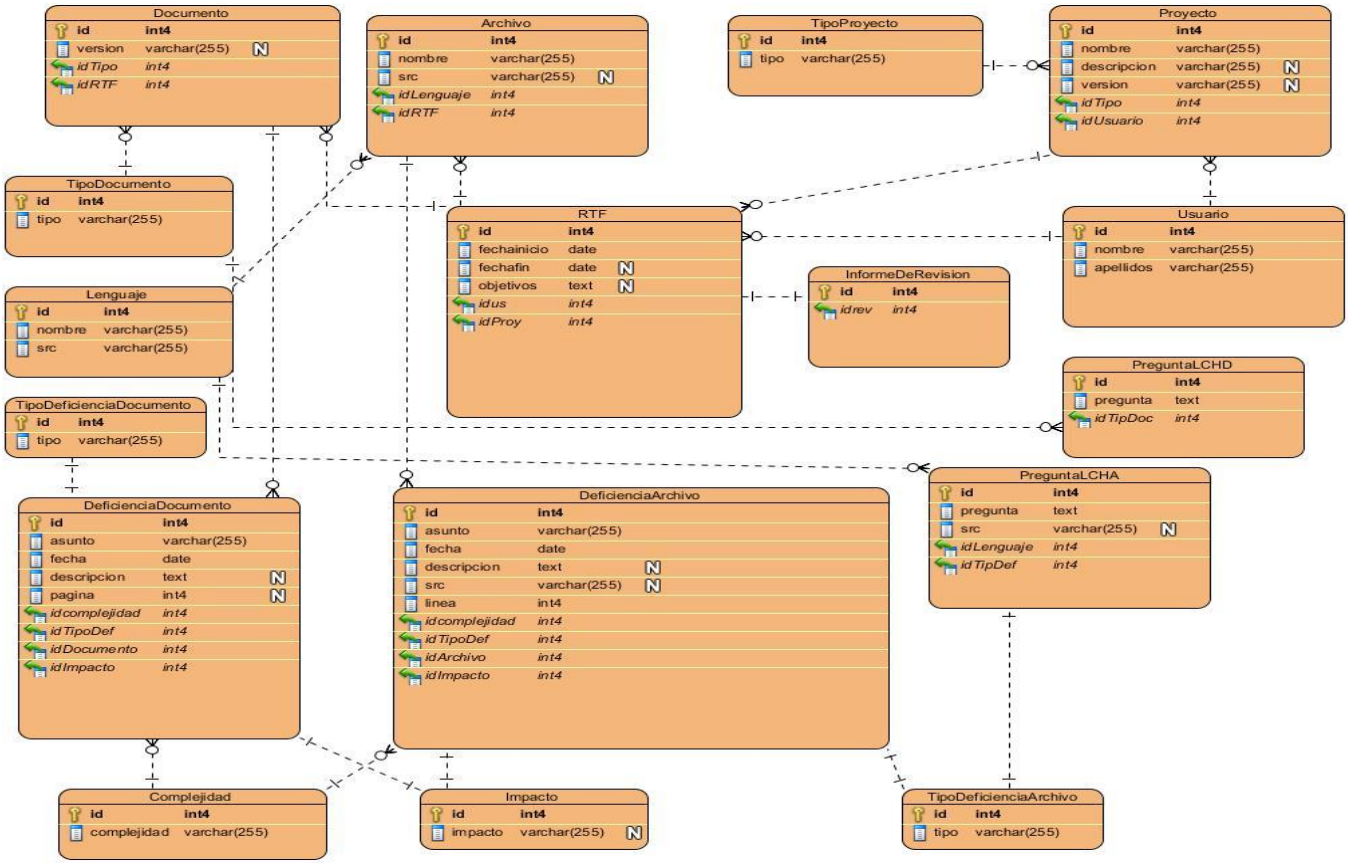


Figura 6. Modelo de datos

Conclusiones del capítulo

Una vez elaborada la propuesta de solución y terminadas las etapas de exploración, planificación y diseño de la metodología XP se concluye que:

- Las listas de chequeo elaboradas tienen los elementos fundamentales para asegurar la calidad del código y de los documentos que se generan en la etapa de Implementación en el centro FORTES.
- La herramienta elaborada cuenta con las características necesarias para realizar el proceso de RTF en el área de Implementación en el centro FORTES.
- Para la implementación de los veinticuatro requisitos del sistema se necesitan veintidós semanas de trabajo por los desarrolladores.

Capítulo 3. Implementación y prueba del sistema

Introducción

En el presente capítulo se exponen aspectos relacionados con la implementación del *software*. Se describen tareas de ingeniería generadas en cada una de las iteraciones y las pruebas realizadas a la herramienta; dichas pruebas permiten a los desarrolladores conseguir que el producto tenga un alto grado de fiabilidad que se caracterice por la ausencia de errores.

3.1 Implementación

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de *software* y a la aplicación disciplinada de las siguientes prácticas (27).

- El juego de la planificación: hay una comunicación frecuente entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
- Entregas pequeñas: producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más tres meses.
- Metáfora: el sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
- Diseño simple: se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

- Pruebas: la producción de código está dirigida por las pruebas unitarias. Estas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
- Refactorización (*Refactoring*): es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
- Programación en parejas: toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores).
- Propiedad colectiva del código: cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- Integración continua: cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- Cuarenta horas por semana: se debe trabajar un máximo de cuarenta horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
- Cliente in-situ: el cliente tiene que estar presente y disponible todo el tiempo para el equipo. Este es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
- Estándares de programación: XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

3.1.1 Patrones de diseño

Un patrón es una forma de dar solución a un problema, con un nombre y que es aplicable a otros contextos, cuenta con una sugerencia sobre la manera de usarlo en situaciones nuevas (55).

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de *software*. Brindan una solución ya probada y documentada a problemas que están sujetos a contextos similares (56).

Los patrones GRASP representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. GRASP es el acrónimo para *General Responsibility Assignment Software Patterns* (Patrones Generales de *Software* para Asignar Responsabilidades). Los básicos son (57):

Creador

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

La nueva instancia deberá ser creada por la clase que (56):

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

El sistema cuenta con diferentes clases controladoras (RtfController.php, ProyectoController.php, entre otras) las cuales contienen las acciones o funciones que hacen al sistema funcional. En estas clases las acciones se encargan de crear los objetos de las clases que representan las entidades, evidenciando de este modo que cada clase controladora es el "creador" de las entidades. La figura 7 muestra un fragmento de uno de los controladores.

```php
*/
class RtfController extends Controller {

    /**
     * Lists all Rtf entities.
     *
     */
    public function indexAction($id) {
        $em = $this->getDoctrine()->getManager();

        $entities = $em->getRepository('TesisHerramientaBundle:Rtf')->findBy(array('idproy' => $id));
        $entidadProyecto = $em->getRepository('TesisHerramientaBundle:Proyecto')->findOneBy(array('id' => $id));
        $deleteForm = $this->createDeleteForm(-1);
        return $this->render('TesisHerramientaBundle:Rtf:index.html.twig', array(
                    'entities' => $entities,
                    'id' => $id,
                    'proyecto' => $entidadProyecto,
                    'delete_form' => $deleteForm->createView(),
        ));
    }
```

Figura 7. RtfController

Experto

Permite asignar una responsabilidad al experto en información, es decir, la clase que cuenta con la información necesaria para cumplir la responsabilidad (56).

Es uno de los patrones que más se utiliza cuando se trabaja con Symfony, debido a que este *framework* utiliza el ORM (mapeo objeto-relacional) Doctrine para mapear la base de datos. Symfony utiliza esta biblioteca para realizar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla que representan. La figura 8 muestra un ejemplo del uso de esta.

```
public function showAction($id) {
    $em = $this->getDoctrine()->getManager();

    $entity = $em->getRepository('TesisHerramientaBundle:Preguntalchd')->find($id);

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

    $deleteForm = $this->createDeleteForm($id);

    return $this->render('TesisHerramientaBundle:Preguntalchd:show.html.twig', array(
                'entity' => $entity,
                'delete_form' => $deleteForm->createView(),));
}
```

Figura 8.Uso del ORM Doctrine

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 realicen un trabajo enorme. Este patrón mejora la calidad y facilidad del diseño, genera un bajo acoplamiento y promueve la reutilización (56):

Symfony permite asignar responsabilidades con una alta cohesión, por ejemplo la clase *Controller* tiene la responsabilidad de definir las varias acciones, es decir está formada por diferentes funcionalidades que se encuentran estrechamente relacionadas proporcionando que el *software* sea flexible frente a grandes cambios. La figura 9 muestra un fragmento de esta clase.

```php
class Controller extends ContainerAware
{
    /**
     * Generates a URL from the given parameters.
     *
     * @param string         $route         The name of the route
     * @param mixed          $parameters    An array of parameters
     * @param Boolean|string $referenceType The type of reference (one of the constants in UrlGeneratorInterface)
     *
     * @return string The generated URL
     *
     * @see UrlGeneratorInterface
     */
    public function generateUrl($route, $parameters = array(), $referenceType = UrlGeneratorInterface::ABSOLUTE_P
    {
        return $this->container->get('router')->generate($route, $parameters, $referenceType);
    }
```

Figura 9. Clase Controller

Bajo acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas otras. Acoplamiento alto significa que una clase recurre a muchas. El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios y que aumentan la oportunidad de una mayor productividad (56).

En Symfony la clase *Controller* hereda solamente de *BaseController* para lograr un bajo acoplamiento de clases.

Controlador

La mayor parte de los sistemas reciben eventos de entrada externa. En estos casos hay que elegir controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas en la elección de los controladores de eventos. Su utilización propicia que las operaciones del sistema se manejen en la capa de dominio de los objetos y no en la de presentación (56).

Symfony favorece a la utilización de este patrón debido a que las peticiones son manejadas por el controlador frontal, implica que todas las solicitudes son dirigidas a un script PHP que se encarga de instanciar al controlador frontal y este determinará qué controlador se debe ejecutar. La figura 10 muestra el controlador frontal de la propuesta de solución.

```php
<?php

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Debug\Debug;

// If you don't want to setup permissions the proper way, just uncomment the following PHP line
// read http://symfony.com/doc/current/book/installation.html#configuration-and-setup for more information
//umask(0000);

// This check prevents access to debug front controllers that are deployed by accident to production servers.
// Feel free to remove this, extend it, or make something more sophisticated.
if (isset($_SERVER['HTTP_CLIENT_IP'])
    || isset($_SERVER['HTTP_X_FORWARDED_FOR'])
    || !in_array(@$_SERVER['REMOTE_ADDR'], array('127.0.0.1', 'fe80::1', '::1'))
) {
    header('HTTP/1.0 403 Forbidden');
    exit('You are not allowed to access this file. Check '.basename(__FILE__).' for more information.');
}

$loader = require_once __DIR__.'/../app/bootstrap.php.cache';
Debug::enable();

require_once __DIR__.'/../app/AppKernel.php';

$kernel = new AppKernel('dev', true);
$kernel->loadClassCache();
$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);
```

Figura 10. Controlador frontal

3.1.2 Tareas de ingeniería

En la fase de Implementación se hace necesario desglosar los requisitos funcionales en tareas con el objetivo de facilitar el trabajo, por lo que durante el transcurso de cada una de las iteraciones se planifican las que se realizarán para dar solución a las historias de usuario que se derivan de estos. Después de realizar una revisión del plan de iteraciones, se empiezan a descomponer los requisitos funcionales en

tareas, asignando dos personas como responsables del desarrollo de cada una de estas, donde cada una corresponde a un período de uno a tres días de desarrollo.

En la primera iteración se desarrollan las HU de los requisitos del sistema directamente con el proceso de RTF.

Tabla 14. Primera iteración

Iteración	Historia de usuario	Duración
1	Gestionar proyecto	½
	Gestionar archivo	½
	Gestionar documento	½
	Gestionar RTF	¾
	Gestionar deficiencia de archivo	½
	Gestionar deficiencia de documento	½

A continuación se muestra una tarea de ingeniería de esta iteración el resto se encuentra en los anexos (ver anexo 4).

Tabla 15. Tarea de ingeniería #1

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 1
Nombre de la tarea: Implementar la funcionalidad gestionar proyecto.	
Tipo de tarea: Desarrollo	
Fecha inicio: 2/3/2015	**Fecha fin:** 5/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un proyecto en la base de datos del sistema.	

En la segunda iteración se desarrollan las HU de los requisitos del sistema referentes a la administración del sistema.

Tabla 16. Segunda iteración

Iteración	Historia de usuario	Duración
2	Gestionar usuario	¾
	Gestionar impacto	½
	Gestionar tipo de proyecto	½
	Gestionar tipo de documento	½
	Gestionar lenguaje de programación	½
	Gestionar tipo de deficiencia de archivo	½
	Gestionar tipo de deficiencia de documento	½
	Gestionar complejidad de deficiencia	½
	Gestionar lista de chequeo de documento	½
	Gestionar lista de chequeo de archivo	½

A continuación se muestra una tarea de ingeniería de esta iteración el resto se encuentra en los anexos (ver anexo 4).

Tabla 17. Tarea de ingeniería #7

Tarea de Ingeniería	
Número de la tarea: 7	**Número de la HU:** 7
Nombre de la tarea: Implementar la funcionalidad gestionar usuario.	
Tipo de tarea: Desarrollo	
Fecha inicio: 20/3/2015	**Fecha fin:** 23/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite registrar, mostrar y eliminar un usuario.	

En la tercera iteración se desarrollan las HU de los requisitos del sistema referentes a las funcionalidades estadísticas y la funcionalidad que permite detectar deficiencias de forma automática.

Tabla 18. Tercera iteración

Iteración	Historia de usuario	Duración
3	Graficar deficiencias por proyecto por tipo de componente	1
	Graficar cantidad de deficiencias por tipo de componente	1
	Graficar cantidad de deficiencias por lenguaje por tipo de deficiencia	1
	Graficar cantidad promedio de deficiencias por lenguaje por componente	1
	Generar tabla resumen de RTF	1
	Graficar resultado de RTF	1
	Generar Informe de RTF	1
	Mostrar posibles deficiencias identificadas en archivo de lenguaje PHP.	3

A continuación se muestra una tarea de ingeniería de esta iteración el resto se encuentra en los anexos (ver anexo 4).

Tabla 19. Tarea de ingeniería #17

Tarea de Ingeniería	
Número de la tarea: 17	**Número de la HU:** 17
Nombre de la tarea: Implementar la funcionalidad graficar deficiencias por proyecto por tipo de componente.	
Tipo de tarea: Desarrollo	
Fecha inicio: 24/4/2015	Fecha fin: 27/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite mostrar una gráfica que muestra las deficiencias por proyecto por tipo de componente.	

3.2 Pruebas

La verificación del *software* se define como actividades que se realizan para comprobar si las salidas en las diferentes etapas y/o subprocesos de desarrollo (código y artefactos) cumplen con las condiciones o los requerimientos impuestos sobre ellos en las entradas por las etapas previas. Constatan que el producto de cada etapa del proyecto es adecuado, completo, consistente y que está acorde a los requerimientos establecidos en las entradas. Esto quiere decir que con las verificaciones se asegura que el producto satisface los requisitos especificados para el diseño en el transcurso de todas las actividades realizadas durante el ciclo de vida del desarrollo (58).

3.2.1 Pruebas unitarias

Son una forma de comprobar el correcto funcionamiento de un módulo de código. Asegura que cada uno funcione correctamente por separado. La idea es escribir casos de prueba para cada función no trivial o método, de forma que cada caso sea independiente del resto. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas fomentan el cambio, simplifica la integración, documenta el código, separación de la interfaz y la implementación y los errores están más acotados y son más fáciles de localizar (59).

Symfony2 brinda su propio sistema de pruebas. Estas verifican que un método se comporta como debería, sin tener en cuenta su entorno. Se realizaron un total de tres iteraciones de pruebas a la herramienta, fueron encontrados 7 errores, todos de ellos posteriormente resueltos. La figura 11 muestra un ejemplo de una prueba unitaria realizada.

```php
public function testBuscarRTF() {

    $arrayIDProy1 = array(
        'fecha' => '2014-09-09',
        'objetivos' => "Buscar errores en el codigo fuente",
        'idproy' => 1,
    );

    $result1 = ExpresionGroupTool::buscarRTF(1);
    $this->assertEquals($arrayIDProy1, $result1);

    $result = ExpresionGroupTool::buscarRTF(4);
    $this->assertEquals(null, $result);
}
```

Figura 11. Ejemplo de prueba unitaria realizada

3.2.2 Pruebas de aceptación

El uso de cualquier producto de *software* tiene que estar justificado por las ventajas que ofrece. Sin embargo, antes de empezar a usarlo es muy difícil determinar si sus ventajas realmente justifican su uso. El mejor instrumento para esta determinación es la llamada prueba de aceptación. En estas se evalúa el grado de calidad del producto con relación a todos los aspectos relevantes para que el uso del producto se justifique. Para eliminar la influencia de conflictos de intereses y para que sea lo más objetiva posible, la prueba de aceptación nunca debería ser responsabilidad de los ingenieros de *software* que han desarrollado el producto.

Para la preparación, la ejecución y la evaluación de la prueba de aceptación no es necesario contar con un conocimiento avanzado de informática. Sin embargo, un conocimiento amplio de métodos y técnicas de prueba y de la gestión de la calidad en general facilita esta labor. La persona o el equipo adecuado para llevar a cabo las pruebas de aceptación disponen de estos conocimientos y además son capaces de interpretar los requerimientos especificados por los futuros usuarios del sistema de *software* en cuestión (60). A continuación se muestran algunos de los casos de pruebas, el resto se encuentra en los anexos (ver anexo 3).

Tabla 20. Prueba de aceptación #1

Caso de prueba de aceptación	
Código: H1_P1	**Historia de usuario:** 1
Nombre: Registrar proyecto	
Descripción: Probar la funcionalidad que permite registrar un nuevo proyecto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar proyecto¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado de los proyectos registrados en el cual debe estar el insertado en el momento.	
Evaluación de la prueba: Satisfactoria	

Tabla 21. Prueba de aceptación #2

Caso de prueba de aceptación	
Código: H1_P2	**Historia de usuario:** 1
Nombre: Editar proyecto	
Descripción: Probar la funcionalidad que permite editar un proyecto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar proyecto¨ a un proyecto de la lista, se llenan los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado de los proyectos registrados en el cual debe haberse modificado el proyecto.	
Evaluación de la prueba: Satisfactoria	

Tabla 22. Prueba de aceptación #3

Caso de prueba de aceptación	
Código: H1_P3	**Historia de usuario:** 1
Nombre: Eliminar proyecto	
Descripción: Probar la funcionalidad que permite eliminar un proyecto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema.	

Entrada/Pasos de ejecución: Se presiona el botón ¨Eliminar¨ asociado a un proyecto de la lista y se después se confirma la eliminación de proyecto en una ventana en forma de alerta.
Resultados esperados: La herramienta muestra el listado de los proyectos registrados en el cual no aparece el proyecto eliminado.
Evaluación de la prueba: Satisfactoria

Resultado de las pruebas

Se realizaron 3 iteraciones para probar el correcto funcionamiento del sistema. Se detectaron un total de 5 no conformidades significativas de las cuales 2 fueron de funcionalidad y 3 de validación, 12 no significativas, todas estas de ortografía y 15 recomendaciones. Las pruebas aplicadas contribuyeron a mejorar la calidad y las funcionalidades del sistema donde se arrojaron resultados visibles. La figura 12 muestra los resultados anteriormente mencionados.

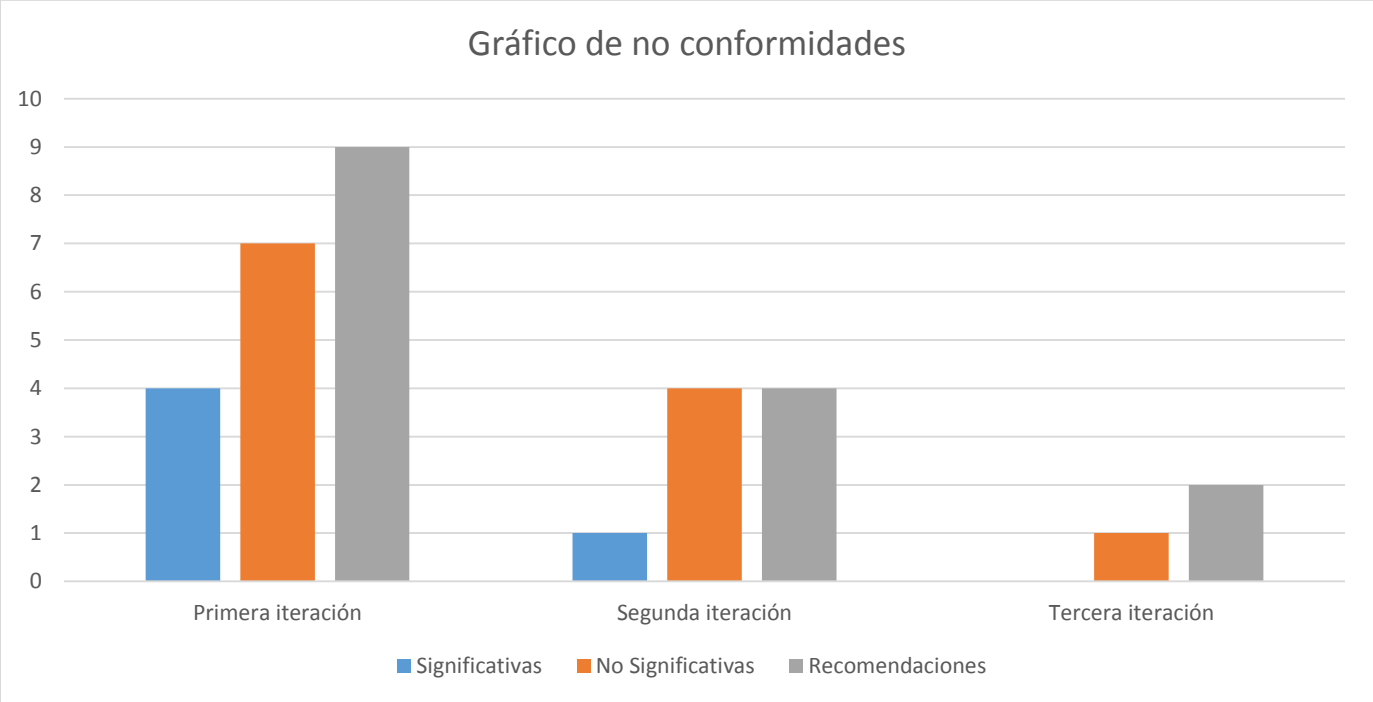


Figura 12. Gráfico de no conformidades

Conclusiones

Después de realizada a implementación y las pruebas de la herramienta se concluye que:

- El tiempo real para la implementación de las funcionalidades estuvo en correspondencia con el que se estimó en la fase de planificación.
- Las pruebas unitarias permitieron obtener un código de calidad.
- Las pruebas de aceptación fueron importantes para que la herramienta tuviese un funcionamiento correcto y de esta forma ayudar al proceso de revisiones del centro FORTES.

Conclusiones generales

Con la realización de este trabajo de diploma se obtuvo una propuesta que da cumplimiento al objetivo general planteado, al lograr desarrollar una herramienta informática que permita agilizar el procedimiento de RTF en el área de Implementación del centro FORTES. Para finalizar se puede concluir que:

- Es importante asegurar la calidad de *software* a través de la realización de revisiones para lograr un producto que se ajuste a las necesidades crecientes de los clientes del mundo contemporáneo.
- La herramienta elaborada cuenta con las características necesarias para realizar el proceso de RTF en el área de Implementación en el centro FORTES, ya que las existentes a pesar de tener una gran calidad, no se ajustan a los estándares del centro.
- La realización de los diferentes tipos de pruebas posibilitaron que la herramienta contara con un correcto funcionamiento, además de tener con un código de calidad y de esta forma poder agilizar el procedimiento de RTF en el centro FORTES.

Recomendaciones

Después de la realización de todo el proceso de investigación y desarrollo de la herramienta informática, se definieron varias recomendaciones necesarias para su mejor funcionamiento y aprovechamiento por parte de quienes la utilicen:

1. Definir estándares de codificación para los demás lenguajes de programación utilizados en el centro FORTES y a partir de estos elaborar listas de chequeo.
2. Extender el uso de la aplicación por los demás centros productivos de la Universidad de las Ciencias Informáticas.

Bibliografía

1. La ausencia de calidad en el *software* genera pérdidas económicas en las empresas. [Citado el: 11 de febrero de 2015.] http://www.computing.es/calidad-*software*/noticias/1071252012701/ausencia-calidad-*software*-genera.1.html.

2. Pressman, Roger S. *Software Engineering A practitioner´s approach.* 7ma.

3. McKinsey & Company. [En línea] octubre de 2012. http://www.mckinsey.com/insights/business_technology/delivering_large-scale_it_projects_on_time_on_budget_and_on_value.

4. Pressman, Roger S. Capítulo 26: Gestión de la calidad. [aut. libro] Roger Pressman. *Software Engineering A practitioner´s approach.*

5. Alvarez de Zayas, Dr. Cs. Carlos. Metodología de la Investigación Científica. [aut. libro] Dr. Cs. Carlos Alvarez de Zayas. *Metodología de la Investigación Científica.* Santiago de Cuba : s.n., 1995.

6. 610-1900, IEEE. *Calidad de software.*

7. 8402-1994, ISO. *Calidad de software.*

8. Pressman, Roger S. *Reflexión de Calidad del Software de Roger Pressman.* 2010.

9. Lección 34 - Revisiones del *Software*. *Unniversidad Nacional Abierta y a Distancia.* [En línea] [Citado el: 8 de febrero de 2015.] http://datateca.unad.edu.co/contenidos/301404/301404_ContenidoEnLinea/leccin_34__revisiones_del_*software*.html.

10. Microsoft. [En línea] http://www.microsoft.com/.

11. MSN Microsoft. [En línea] https://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

12. Naiman, Gabriel. *Las revisiones de código, renovadas y más vigentes que nunca.*

13. National Aeronautics and Space Administration Washington, DC 20546. *SOFTWARE FORMAL INSPECTIONS GUIDEBOOK.*

14. Definición. *Definicion de Manual de Usuario.* [En línea] [Citado el: 10 de febrero de 2015.] http://definicion.de/manual-de-usuario/.

15. Alegsa.com.ar. *¿Cuál es la definicion de Código fuente?* [En línea] [Citado el: 8 de 2 de 2015.] http://www.alegsa.com.ar/Dic/codigo%20fuente.php.

16. ciclodevida*software*. *Lenguajes de programación.* [En línea] [Citado el: 10 de febrero de 2015.] http://ciclodevida*software*.wikispaces.com/Lenguajes+De+Programacion.

17. ciclodevida*software*. *Clasificacion De Los Lenguajes De Programacion.* [En línea] [Citado el: 10 de febrero de 2015.] http://ciclodevida*software*.wikispaces.com/Clasificacion+De+Los+Lenguajes+De+Programacion.

18. Díaz, Indira Pérez. *Procedimiento de Revisiones de Software.* La Habana : s.n., 2012.

19. *Procedimiento para realizar revisiones técnicas formales a la actividad productica en la UCI.*

20. Maulini R, Mauro. Desarrollo y Seguridad de Aplicaciones Web y Móviles. [En línea] http://tecnologiasweb.blogspot.com/2010/11/owasp-code-crawler.html.

21. García, Luis Miguel. Review Board: herramienta web revisión de código | Un poco de Java. [En línea] 11 de Julio de 2012. https://unpocodejava.wordpress.com/2012/07/11/review-board-herramienta-web-revision-de-codigo/.

22. PMD y la calidad estática del código | Marco de Desarrollo de la Junta de Andalucía. [En línea] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/374.

23. Federico Caro. Sonar: Medidad de la calidad de tu código. [En línea] 16 de Septiembre de 2010. http://www.paradigmatecnologico.com/blog/sonar-medida-de-la-calidad-de-tu-codigo/.

24. métodos ágiles vs métodos tradicionales. *Metodologías ágiles.* [En línea] 8 de febrero de 2015. http://metodologiasagiles.wikispaces.com/metodos+agiles+vs+metodos+tradicionales.

25. Bolivariana, Universidad Unión. *METODOLOGIAS AGILES Proceso Unificado Ágil (AUP).* La Paz : s.n. pág. 7.

26. Informáticas, Universidad de las Ciencias. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana : s.n.

27. Penadés, Patricio Letelier y Mª Carmen. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* Valencia : s.n.

28. José H. Canós, Patricio Letelier y María del Carmen Penadés. *Métodologías Ágiles en el Desarrollo de Software.* Valencia : s.n.

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

30. Definición de Lenguaje de Programación. *Definición.de.* [En línea] [Citado el: 9 de febrero de 2015.] http://definicion.de/lenguaje-de-programacion/.

31. 12 ventajas de HTML5. *12 ventajas de HTML5.* [En línea] [Citado el: 15 de 2 de 2015.] http://www.go-movil.es/index.php/12-ventajas-del-html5/.

32. 12 ventajas del HTML5. *Movil.es.* [En línea] [Citado el: 10 de febrero de 2015.] http://www.go-movil.es/index.php/12-ventajas-del-html5/.

33. Tecnología Innovadora. *¿CUÁLES SON LAS VENTAJAS Y DESVENTAJAS DE CSS3?* [En línea] 14 de mayo de 2012. [Citado el: 12 de 2 de 2015.] http://tecnologiainnovadoraunad.blogspot.com/2012/05/cuales-son-las-ventajas-y-desventajas_14.html.

34. Introducción a Javascript(Teoria). *Codigo Programación.* [En línea] [Citado el: 11 de febrero de 2015.] http://codigoprogramacion.com/cursos/javascript/88-introduccion-javascript.html.

35. Definición de XML. *Definición.de.* [En línea] [Citado el: 9 de febrero de 2105.] http://definicion.de/xml/.

36. Ventajas y Desventajas de XML. *FUNDAMENTOSDEXML.* [En línea] [Citado el: 11 de febrero de 2015.] https://fundamentosdexml.wordpress.com/2012/04/19/ventajas-y-desventajas-de-xml-2/.

37. Definición de UML. *DICCIONARIO DE INFORMÁTICA Y TECNOLOGÍA.* [En línea] [Citado el: 11 de febrero de 2015.] http://www.alegsa.com.ar/Dic/uml.php.

38. PHP: Hypertext Preprocessor. [En línea] [Citado el: 8 de febrero de 2015.] http://php.net/.

39. jordisan.net. *¿Qué es un framework?* [En línea] 29 de septiembre de 2006. [Citado el: 8 de febrero de 2015.] http://jordisan.net/blog/2006/que-es-un-framework/.

40. *Curso Básico de Symfony 2.* 2012.

41. ¿Por qué Symfony? *Blog de Javier Negreira.* [En línea] [Citado el: 11 de febrero de 2015.] http://javiernegreira.com/blog/2014/09/por-que-symfony/.

42. Herramientas CASE. *Slideshare.* [En línea] [Citado el: 11 de febrero de 2015.] http://es.slideshare.net/guestf131a9/herramientas-case.

43. Visual Paradigm for UML - Libere las revisiones de la transferencia directa y del *software* | CNET Download.com. [En línea] http://descargar.cnet.com/Visual-Paradigm-for-UML/3000-2247_4-42700.html.

44. Definición IDE Enciclopedia Proyecto AjpdSoft. [En línea] [Citado el: 8 de febrero de 2015.] http://www.ajpdsoft.com/modules.php?name=Encyclopedia&op=content&tid=1051.

45. JetBrains. [En línea] febrero de 2015. https://www.jetbrains.com/phpstorm/.

46. Domínguez-Dorado, M. *Todo Programación.* Madrid : Editorial Iberprensa, 2005.

47. NetBeans IDE. [En línea] 2015. https://netbeans.org/.

48. Classora. [En línea] [Citado el: 7 de febrero de 2015.] http://es.classora.com/reports/x46901/ranking-de-las-mejores-bases-de-datos-actuales.

49. PostgreSQL-es. *PostgreSQL-es.* [En línea] 2 de 10 de 2010. http://www.postgresql.org.es/sobre_postgresql.

50. Cliente grafico: pgAdmin3. *Dataprix.* [En línea] [Citado el: 11 de febrero de 2015.] http://www.dataprix.com/8-cliente-grafico-pgadmin3.

51. pgadmin. *pgadmin.* [En línea] [Citado el: 11 de febrero de 2015.] http://www.pgadmin.org/.

52. Definición de Servidor de base de datos. *Alegsa.* [En línea] [Citado el: 11 de febrero de 2015.] http://www.alegsa.com.ar/Dic/servidor%20de%20base%20de%20datos.php.

53. What is the Apache HTTP Server Project? *Apache.* [En línea] [Citado el: 11 de febrero de 2015.] http://httpd.apache.org/ABOUT_APACHE.html.

54. Eguiluz, Javier. *Desarrollo web ágil con Symfony 2.*

55. Larman, Craig. *UML y patrones.* 1999.

56. ¿Qué es un Patrón de Diseño? *Microsoft.* [En línea] [Citado el: 16 de febrero de 2015.] https://msdn.microsoft.com/es-es/library/bb972240.aspx.

57. Gamma, Erich. *Design Patterns. Elements of Reusable Object-Oriented Software.* 1995.

58. Blanco Llano, Javier y Rodriguez Hernandez, Aida. *REVISIÓN, VERIFICACIÓN Y VALIDACIÓN.* 2011.

59. Enfoque para pruebas de unidad basado en la generación aleatoria de objetos. *SEDICI.* [En línea] [Citado el: 12 de febrero de 2015.] http://sedici.unlp.edu.ar/handle/10915/34969.

60. pruebasde*software*. *La prueba de aceptación.* [En línea] [Citado el: 12 de febrero de 2015.] http://pruebas de *software*.com/pruebadeaceptacion.htm.

Anexos

Anexo 1: Historias de usuario

Tabla 23. Historia de usuario #2

Historia de Usuario	
Número: 2	**Nombre:** Gestionar archivo
Usuario: Usuario normal	**Iteración:** 1
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar un archivo para su posterior revisión.	

Tabla 24. Historia de usuario #3

Historia de Usuario	
Número: 3	**Nombre:** Gestionar documento
Usuario: Usuario normal	**Iteración:** 1
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar un documento para su posterior revisión.	

Tabla 25. Historia de usuario #4

Historia de Usuario	
Número: 4	**Nombre:** Gestionar RTF
Usuario: Usuario normal	**Iteración:** 1
Prioridad en el negocio: Alta	**Puntos estimados:** 3/4
Riesgo en el desarrollo: Alto	**Puntos reales:** 3/4
Descripción: La aplicación debe permitir registrar, mostrar y eliminar una RTF.	

Tabla 26. Historia de usuario #5

Historia de Usuario	
Número: 5	**Nombre:** Gestionar deficiencia de archivo
Usuario: Usuario normal	**Iteración:** 1
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2

Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y eliminar una deficiencia asociada a un archivo durante el desarrollo de una RTF.	

Tabla 27. Historia de usuario #6

Historia de Usuario	
Número: 6	**Nombre:** Gestionar deficiencia de documento
Usuario: Usuario normal	**Iteración:** 1
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y eliminar una deficiencia asociada a un documento durante el desarrollo de una RTF.	

Tabla 28. Historia de usuario #7

Historia de Usuario	
Número: 7	**Nombre:** Gestionar usuario
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 3/4
Riesgo en el desarrollo: Alto	**Puntos reales:** 3/4
Descripción: La aplicación debe permitir registrar, eliminar, mostrar y autenticar un usuario del sistema.	

Tabla 29. Historia de usuario #8

Historia de Usuario	
Número: 8	**Nombre:** Gestionar impacto
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y editar un tipo de impacto.	

Tabla 30. Historia de usuario #12

Historia de Usuario	
Número: 12	Nombre: Gestionar tipo de deficiencia de

	archivo
Usuario: Administrador	Iteración: 2
Prioridad en el negocio: Alta	Puntos estimados: 1/2
Riesgo en el desarrollo: Alto	Puntos reales: 1/2
Descripción: La aplicación debe permitir registrar y editar los tipos de deficiencias de archivo que contiene para clasificar el sistema.	

Tabla 31. Historia de usuario #13

Historia de Usuario	
Número: 13	**Nombre:** Gestionar tipo de deficiencia de documento
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y editar los tipos de deficiencia de documentos que contiene para clasificar el sistema.	

Tabla 32. Historia de usuario #14

Historia de Usuario	
Número: 14	**Nombre:** Gestionar complejidad de deficiencia
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar y editar los tipos de complejidad que contiene para clasificar el sistema.	

Tabla 33. Historia de usuario #15

Historia de Usuario	
Número: 15	**Nombre:** Gestionar lista de chequeo de documento
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2

Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar, eliminar y editar las preguntas de la lista de chequeo de los documentos.	

Tabla 34. Historia de usuario #16

Historia de Usuario	
Número: 16	**Nombre:** Gestionar lista de chequeo para archivo
Usuario: Administrador	**Iteración:** 2
Prioridad en el negocio: Alta	**Puntos estimados:** 1/2
Riesgo en el desarrollo: Alto	**Puntos reales:** 1/2
Descripción: La aplicación debe permitir registrar, eliminar y editar las preguntas de la lista de chequeo de los archivos.	

Tabla 35. Historia de usuario #17

Historia de Usuario	
Número: 17	**Nombre:** Graficar cantidad de deficiencias por proyecto por tipo de componente revisado
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Media	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe permitir mostrar una gráfica donde se muestren los proyectos registrados con la cantidad de deficiencias identificadas a cada uno de los tipos de componentes que tiene el sistema.	

Tabla 36. Historia de usuario #20

Historia de Usuario	
Número: 20	**Nombre:** Graficar cantidad promedio de deficiencias por lenguaje por componente
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Media	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe permitir mostrar una gráfica con la cantidad promedio de	

deficiencias identificadas en cada uno de los lenguajes registrados en el sistema.

Tabla 37. Historia de usuario #21

Historia de Usuario	
Número: 21	**Nombre:** Generar tabla resumen de RTF
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Alta	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe generar una tabla resumen después de terminada una RTF.	

Tabla 38. Historia de usuario #22

Historia de Usuario	
Número: 22	**Nombre:** Graficar resultados de RTF
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Media	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe permitir mostrar una gráfica con el resumen de los resultados de una RTF.	

Tabla 39. Historia de usuario #23

Historia de Usuario	
Número: 23	**Nombre:** Generar Informe de RTF
Usuario: Usuario normal	**Iteración:** 3
Prioridad en el negocio: Alta	**Puntos estimados:** 1
Riesgo en el desarrollo: Medio	**Puntos reales:** 1
Descripción: La aplicación debe permitir generar un informe en formato PDF con los resultados más significativos de la RTF.	

Anexo 2: Tarjetas CRC

Tabla 40. Tarjeta CRC #3

Clase: Proyecto	
Responsabilidades**:**	Colaboradores:

➢ Obtener nombre. ➢ Modificar nombre. ➢ Obtener descripción. ➢ Modificar descripción. ➢ Obtener versión. ➢ Modificar versión. ➢ Obtener el identificador del tipo de proyecto. ➢ Modificar el identificador del tipo de proyecto. ➢ Obtener el identificador del usuario que lo registró. ➢ Modificar el identificador del usuario que lo registró.	➢ Tipoproyecto ➢ Rtf ➢ Usuario

Tabla 41. Tarjeta CRC #4

Clase: Archivo	
Responsabilidades**:**	Colaboradores:
➢ Obtener el nombre. ➢ Modificar el nombre. ➢ Obtener el identificador del lenguaje de programación. ➢ Modificar el identificador del lenguaje de programación. ➢ Obtener el identificador de la RTF. ➢ Modificar el identificador de la RTF.	➢ Rtf ➢ Deficienciaarchivo ➢ Lenguaje

Tabla 42. Tarjeta CRC #5

Clase: Documento	
Responsabilidades**:**	Colaboradores:
➢ Obtener la versión ➢ Modificar la versión ➢ Obtener el identificador del tipo de documento. ➢ Modificar el identificador del tipo de documento.	➢ Tipodocumento ➢ Deficienciadocumento

➢ Obtener el identificador de la RTF. ➢ Modificar el identificador de la RTF.	

Tabla 43. Tarjeta CRC #6

Clase: Deficiencia de documento	
Responsabilidades**:**	Colaboradores:
➢ Obtener asunto de deficiencia. ➢ Modificar asunto de deficiencia ➢ Obtener fecha de registro. ➢ Modificar fecha de registro. ➢ Obtener descripción. ➢ Modificar descripción. ➢ Obtener ruta de la foto. ➢ Modificar ruta de la foto. ➢ Obtener página. ➢ Modificar página. ➢ Obtener el identificador de complejidad. ➢ Modificar el identificador de la complejidad. ➢ Obtener el identificador del tipo de deficiencia. ➢ Modificar el identificador del tipo d deficiencia. ➢ Obtener el identificador del documento. ➢ Modificar el identificador del documento. ➢ Obtener el identificador del estado de solución. ➢ Modificar el identificador del estado de solución. ➢ Obtener el identificador del impacto. ➢ Modificar el identificador del impacto.	➢ Documento ➢ Tipodeficienciadocumento ➢ Complejidad ➢ Impacto

Tabla 44. Tarjeta CRC #7

Clase: Usuario	
Responsabilidades**:**	Colaboradores:
➢ Obtener nombre.	➢ Proyecto

➢ Modificar nombre. ➢ Obtener apellidos. ➢ Modificar apellidos.	

Tabla 45. Tarjeta CRC #8

Clase: Tipo de proyecto	
Responsabilidades**:**	Colaboradores:
➢ Obtener tipo de proyecto. ➢ Modificar tipo de proyecto.	➢ Proyecto

Tabla 46. Tarjeta CRC #9

Clase: Tipo de documento	
Responsabilidades**:**	Colaboradores:
➢ Obtener tipo de proyecto. ➢ Modificar tipo de proyecto.	➢ Documento ➢ Preguntalchd

.Tabla 47. Tarjeta CRC #10

Clase: Lenguaje de programación	
Responsabilidades**:**	Colaboradores:
➢ Obtener nombre. ➢ Modificar nombre. ➢ Obtener ruta del logo. ➢ Modificar ruta del logo.	➢ Archivo ➢ Preguntalcha

Tabla 48. Tarjeta CRC #11

Clase: Tipo de deficiencia de archivo	
Responsabilidades**:**	Colaboradores:
➢ Obtener tipo de deficiencia. ➢ Modificar tipo de deficiencia.	➢ Deficienciaarchivo ➢ Preguntalcha

Tabla 49. Tarjeta CRC #12

Clase: Tipo de deficiencia de documento	
Responsabilidades**:**	Colaboradores:
➢ Obtener tipo de deficiencia. ➢ Modificar tipo de deficiencia.	➢ Deficienciadocumento

Tabla 50. Tarjeta CRC #13

Clase: Complejidad de deficiencia	
Responsabilidades**:**	Colaboradores:
➢ Obtener complejidad. ➢ Modificar complejidad.	➢ Deficienciadocumento ➢ Deficienciaarchivo

Tabla 51. Tarjeta CRC #14

Clase: Preguntalcha	
Responsabilidades**:**	Colaboradores:
➢ Obtener pregunta. ➢ Modificar pregunta. ➢ Obtener ruta de foto. ➢ Modificar ruta de foto. ➢ Obtener el identificador del lenguaje. ➢ Modificar el identificador del lenguaje. ➢ Obtener el identificador de tipo de deficiencia. ➢ Modificar el identificador de tipo de deficiencia.	➢ Lenguaje ➢ Tipodeficienciaarchivo

Tabla 52. Tarjeta CRC #15

Clase: Preguntalchd	
Responsabilidades**:**	Colaboradores:

➢ Obtener pregunta. ➢ Modificar pregunta. ➢ Obtener el identificador del tipo de documento. ➢ Modificar el identificador del tipo de documento.	➢ Tipodeficienciadocumento

Tabla 53. Tarjeta CRC #16

Clase: Impacto	
Responsabilidades**:**	Colaboradores:
➢ Obtener impacto. ➢ Modificar impacto.	➢ Deficienciaarchivo ➢ Deficienciadocumento

Tabla 54. Tarjeta CRC #17

Clase: RTF	
Responsabilidades**:**	Colaboradores:
➢ Obtener fecha de inicio. ➢ Modificar fecha de inicio. ➢ Obtener fecha de terminación. ➢ Modificar fecha de terminación. ➢ Obtener el objetivo. ➢ Modificar el objetivo. ➢ Obtener el identificador del usuario que la registró. ➢ Modificar el identificador del usuario que la registró. ➢ Obtener el identificador del proyecto al cual pertenece. ➢ Modificar el identificador del proyecto al cual pertenece.	➢ Usuario ➢ Proyecto

Anexo 3. Casos de prueba de aceptación

Tabla 55. Prueba de aceptación #4

Caso de prueba de aceptación	
Código: H2_P1	**Historia de usuario:** 2
Nombre: Registrar archivo	
Descripción: Probar la funcionalidad que permite registrar un nuevo archivo en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema y debe estar ubicado en la vista de la RTF a la cual se le va a registrar el archivo.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Archivo¨ en la vista de la RTF, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta retorna a la vista donde se muestra la RTF a la que pertenece el archivo.	
Evaluación de la prueba: Satisfactoria	

Tabla 56. Prueba de aceptación #5

Caso de prueba de aceptación	
Código: H3_P1	**Historia de usuario:** 3
Nombre: Registrar documento	
Descripción: Probar la funcionalidad que permite registrar un nuevo documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema y debe estar ubicado en la vista de la RTF a la cual se le va a registrar el documento.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Documento¨ en la vista de la RTF, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta retorna a la vista donde se muestra la RTF a la que pertenece el documento.	
Evaluación de la prueba: Satisfactoria	

Tabla 57. Prueba de aceptación #6

Caso de prueba de aceptación	
Código: H4_P1	**Historia de usuario:** 4
Nombre: Registrar RTF	

Descripción: Probar la funcionalidad que permite registrar una RTF en la aplicación.
Condiciones de ejecución: Debe estar autenticado en el sistema en la vista del listado de proyectos registrado.
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar RTF¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.
Resultados esperados: La herramienta muestra la vista de la RTF registrada.
Evaluación de la prueba: Satisfactoria

Tabla 58. Prueba de aceptación #7

Caso de prueba de aceptación	
Código: H4_P3	**Historia de usuario:** 4
Nombre: Eliminar RTF	
Descripción: Probar la funcionalidad que permite eliminar una RTF en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en la vista donde se muestran todas las RTF asociadas a un proyecto.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Eliminar¨ asociado a una RTF de la lista, y después se confirma la eliminación de la RTF en una ventana en forma de alerta.	
Resultados esperados: La herramienta muestra el listado de las RTF asociadas al proyecto en el cual no aparece la RTF eliminada.	
Evaluación de la prueba: Satisfactoria	

Tabla 59. Prueba de aceptación #8

Caso de prueba de aceptación	
Código: H5_P1	**Historia de usuario:** 5
Nombre: Registrar deficiencia de archivo	
Descripción: Probar la funcionalidad que permite registrar una deficiencia de archivo en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema y estar ubicado en la vista de revisión de archivo.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Añadir¨ en la lista de chequeo asociada al archivo tipo de lenguaje del archivo, se llena el formulario de la deficiencia y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta retorna a la vista de revisión de archivo.	
Evaluación de la prueba: Satisfactoria	

Tabla 60. Prueba de aceptación #9

Caso de prueba de aceptación	
Código: H5_P2	**Historia de usuario:** 5
Nombre: Eliminar deficiencia de archivo	
Descripción: Probar la funcionalidad que permite eliminar una deficiencia de archivo en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema y estar ubicado en la vista que muestra todas las deficiencias asociadas a un archivo.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Eliminar¨ asociado a una deficiencia de archivo de la lista y después se confirma la eliminación de la deficiencia en una ventana en forma de alerta.	
Resultados esperados: La herramienta muestra el listado de deficiencias asociadas al archivo en el cual no aparece la deficiencia eliminada.	
Evaluación de la prueba: Satisfactoria	

Tabla 61. Prueba de aceptación #10

Caso de prueba de aceptación	
Código: H6_P1	**Historia de usuario:** 6
Nombre: Registrar deficiencia de documento	
Descripción: Probar la funcionalidad que permite registrar una deficiencia de documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema y estar ubicado en la vista de revisión de documento.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar deficiencia¨, se llena el formulario de la deficiencia y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra la lista de chequeo asociada al documento.	
Evaluación de la prueba: Satisfactoria	

Tabla 62. Prueba de aceptación #11

Caso de prueba de aceptación	
Código: H6_P2	**Historia de usuario:** 6
Nombre: Eliminar deficiencia de documento	
Descripción: Probar la funcionalidad que permite eliminar una deficiencia de documento en la aplicación.	

Condiciones de ejecución: Debe estar autenticado en el sistema y estar ubicado en la vista que muestra todas las deficiencias asociadas a un documento.
Entrada/Pasos de ejecución: Se presiona el botón ¨Eliminar¨ asociado a una deficiencia de documento de la lista, y después se confirma la eliminación de la deficiencia en una ventana en forma de alerta.
Resultados esperados: La herramienta muestra el listado de deficiencias asociadas al documento en el cual no aparece la deficiencia eliminada.
Evaluación de la prueba: Satisfactoria

Tabla 63. Prueba de aceptación #12

Caso de prueba de aceptación	
Código: H7_P1	**Historia de usuario:** 7
Nombre: Autenticar usuario	
Descripción: Evaluar la autenticación de un usuario.	
Condiciones de ejecución:	
Entrada/Pasos de ejecución: Se llena el formulario del usuario y luego se presiona el botón ¨Aceptar¨.	
Resultados esperados: La herramienta muestra la interfaz principal.	
Evaluación de la prueba: Satisfactoria	

. Tabla 64. Prueba de aceptación #13

Caso de prueba de aceptación	
Código: H8_P1	**Historia de usuario:** 8
Nombre: Registrar impacto	
Descripción: Probar la funcionalidad que permite registrar un impacto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de impactos registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar impacto¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado los impactos registrados en el cual debe aparecer el añadido.	
Evaluación de la prueba: Satisfactoria	

Tabla 65. Prueba de aceptación #14

Caso de prueba de aceptación	
Código: H8_P2	**Historia de usuario**: 8
Nombre: Editar impacto	
Descripción: Probar la funcionalidad que permite editar un impacto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de impactos registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a un impacto, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado los impactos registrados con el impacto editado actualizado.	
Evaluación de la prueba: Satisfactoria	

Tabla 66. Prueba de aceptación #15

Caso de prueba de aceptación	
Código: H9_P1	**Historia de usuario:** 9
Nombre: Registrar tipo de proyecto	
Descripción: Probar la funcionalidad que permite registrar un tipo de proyecto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de proyecto registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar tipo¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado los tipos de proyecto registrados en el cual debe aparecer el añadido.	
Evaluación de la prueba: Satisfactoria	

Tabla 67. Prueba de aceptación #16

Caso de prueba de aceptación	
Código: H9_P2	**Historia de usuario:** 9
Nombre: Editar tipo de proyecto	
Descripción: Probar la funcionalidad que permite editar un tipo de proyecto en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista	

de tipos de proyecto registrados.
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a un tipo de proyecto, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.
Resultados esperados: La herramienta muestra el listado los tipos de proyecto registrados con el tipo de proyecto editado actualizado.
Evaluación de la prueba: Satisfactoria

Tabla 68. Prueba de aceptación #17

Caso de prueba de aceptación	
Código: H10_P1	**Historia de usuario:** 10
Nombre: Registrar tipo de documento	
Descripción: Probar la funcionalidad que permite registrar un tipo de documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de documentos registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar tipo¨, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado los tipos de documento registrados en el cual debe aparecer el añadido.	
Evaluación de la prueba: Satisfactoria	

Tabla 69. Prueba de aceptación #18

Caso de prueba de aceptación	
Código: H10_P2	Historia de usuario: 10
Nombre: Editar tipo de documento	
Descripción: Probar la funcionalidad que permite editar un tipo de documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de documentos registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a un tipo de documento, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado los tipos de documento registrados en el cual se ha modificado el editado.	
Evaluación de la prueba: Satisfactoria	

Tabla 70. Prueba de aceptación #19

Caso de prueba de aceptación	
Código: H11_P1	**Historia de usuario:** 11
Nombre: Registrar lenguaje de programación	
Descripción: Probar la funcionalidad que permite registrar un lenguaje de programación en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de lenguajes de programación registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar lenguaje¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado de los lenguajes de programación registrados en el cual debe aparecer el añadido.	
Evaluación de la prueba: Satisfactoria	

Tabla 71. Prueba de aceptación #20

Caso de prueba de aceptación	
Código: H11_P2	**Historia de usuario:** 11
Nombre: Editar lenguaje de programación	
Descripción: Probar la funcionalidad que permite editar un lenguaje de programación en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de lenguajes de programación registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a un lenguaje de programación, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado los lenguajes de programación registrados en el cual se ha modificado el editado.	
Evaluación de la prueba: Satisfactoria	

Tabla 72. Prueba de aceptación #21

Caso de prueba de aceptación	
Código: H12_P1	**Historia de usuario:** 12
Nombre: Registrar tipo de deficiencia de archivo	
Descripción: Probar la funcionalidad que permite registrar un tipo de deficiencia de archivo en la	

aplicación.
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de deficiencia de archivo registrados.
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar tipo¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.
Resultados esperados: La herramienta muestra el listado los tipos de deficiencia de archivo registrados en el cual debe aparecer el añadido.
Evaluación de la prueba: Satisfactoria

Tabla 73. Prueba de aceptación #22

Caso de prueba de aceptación	
Código: H12_P2	**Historia de usuario:** 12
Nombre: Editar tipo de deficiencia de archivo	
Descripción: Probar la funcionalidad que permite editar un tipo de deficiencia de archivo en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de deficiencia de archivo registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a una tipo de deficiencia de archivo, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado los tipos de deficiencia de archivo registrados en el cual se ha modificado el editado.	
Evaluación de la prueba: Satisfactoria	

Tabla 74. Prueba de aceptación #23

Caso de prueba de aceptación	
Código: H13_P1	**Historia de usuario:** 13
Nombre: Registrar tipo de deficiencia de documento	
Descripción: Probar la funcionalidad que permite registrar un tipo de deficiencia de documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de deficiencia de documento registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar tipo¨, se llenan los campos del	

formulario y se presiona el botón ¨Registrar¨.
Resultados esperados: La herramienta muestra el listado los tipos de deficiencia de documento registrados en el cual debe aparecer el añadido.
Evaluación de la prueba: Satisfactoria

Tabla 75. Prueba de aceptación #24

Caso de prueba de aceptación	
Código: H13_P2	**Historia de usuario:** 13
Nombre: Editar tipo de deficiencia de documento	
Descripción: Probar la funcionalidad que permite editar un tipo de deficiencia de documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de tipos de deficiencia de documento registrados.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a una tipo de deficiencia de documento, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado los tipos de deficiencia de documento registrados en el cual se ha modificado el editado.	
Evaluación de la prueba: Satisfactoria	

Tabla 76. Prueba de aceptación #25

Caso de prueba de aceptación	
Código: H14_P1	**Historia de usuario:** 14
Nombre: Registrar complejidad	
Descripción: Probar la funcionalidad que permite registrar una complejidad en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de complejidades registradas.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar complejidad¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado complejidades registradas en el cual debe aparecer la añadida.	
Evaluación de la prueba: Satisfactoria	

Tabla 77. Prueba de aceptación #26

Caso de prueba de aceptación	
Código: H14_P2	**Historia de usuario:** 14
Nombre: Editar complejidad	
Descripción: Probar la funcionalidad que permite editar una complejidad en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y abrir la lista de complejidades registradas.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨, se llenan los campos del formulario y se presiona el botón ¨Aceptar¨.	
Resultados esperados: La herramienta muestra el listado complejidades registradas en el cual se ha modificado la editada.	
Evaluación de la prueba: Satisfactoria	

Tabla 78. Prueba de aceptación #27

Caso de prueba de aceptación	
Código: H15_P1	**Historia de usuario:** 15
Nombre: Registrar lista de chequeo de documento	
Descripción: Probar la funcionalidad que permite registrar una lista de chequeo de documento en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador	
Entrada/Pasos de ejecución: Se presiona el botón ¨Registrar pregunta¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado de preguntas de la lista de chequeo registradas en el cual debe aparecer la añadida.	
Evaluación de la prueba: Satisfactoria	

Tabla 79. Prueba de aceptación #28

Caso de prueba de aceptación	
Código: H15_P2	**Historia de usuario:** 15
Nombre: Editar lista de chequeo de documento	
Descripción: Probar la funcionalidad que permite editar una lista de chequeo de documento en la aplicación.	

Condiciones de ejecución: Debe estar autenticado en el sistema como administrador
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a una pregunta, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.
Resultados esperados: La herramienta muestra el listado de preguntas de la lista de chequeo registradas en la cual se ha modificado la editada.
Evaluación de la prueba: Satisfactoria

Tabla 80. Prueba de aceptación #29

Caso de prueba de aceptación	
Código: H15_P2	**Historia de usuario:** 15
Nombre: Eliminar pregunta de lista de chequeo	
Descripción: Probar la funcionalidad que permite eliminar una pregunta de la lista de chequeo.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y estar ubicado en la vista que muestra todas las preguntas de una lista de chequeo de uno de los tipos de documento registrados en el sistema.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Eliminar¨ asociado a una pregunta de la lista de chequeo, y después se confirma la eliminación de la pregunta en una ventana en forma de *model*.	
Resultados esperados: La herramienta muestra el listado de preguntas asociadas a la lista de chequeo de uno de los tipos de documento en el cual no aparece la pregunta eliminada.	
Evaluación de la prueba: Satisfactoria	

Tabla 81. Prueba de aceptación #30

Caso de prueba de aceptación	
Código: H16_P1	**Historia de usuario:** 16
Nombre: Registrar lista de chequeo de archivo	
Descripción: Probar la funcionalidad que permite registrar una pregunta a la lista de chequeo de archivo asociada a un lenguaje en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador	
Entrada/Pasos de ejecución: Se presiona el botón ¨Añadir pregunta¨, se llenan los campos del formulario y se presiona el botón ¨Registrar¨.	
Resultados esperados: La herramienta muestra el listado preguntas de la lista de chequeo asociada al lenguaje registradas, en el cual debe aparecer la añadida.	

Evaluación de la prueba: Satisfactoria

Tabla 82. Prueba de aceptación #31

Caso de prueba de aceptación	
Código: H16_P2	**Historia de usuario:** 16
Nombre: Editar lista de chequeo de archivo	
Descripción: Probar la funcionalidad que permite editar una pregunta de la lista chequeo de archivo asociada a un lenguaje en la aplicación.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y estar ubicado en la vista que muestra las preguntas de la lista de chequeo.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Editar¨ asociado a una pregunta, se modifican los campos del formulario y se presiona el botón ¨Actualizar¨.	
Resultados esperados: La herramienta muestra el listado de preguntas de la lista de chequeo de archivo asociadas a un lenguaje, en el cual se ha modificado la editada.	
Evaluación de la prueba: Satisfactoria	

Tabla 83. Prueba de aceptación #32

Caso de prueba de aceptación	
Código: H16_P2	**Historia de usuario:** 16
Nombre: Eliminar pregunta de lista de chequeo	
Descripción: Probar la funcionalidad que permite eliminar una pregunta de la lista de chequeo.	
Condiciones de ejecución: Debe estar autenticado en el sistema como administrador y estar ubicado en la vista que muestra todas las preguntas de una lista de chequeo asociada a uno de los lenguajes registrados en el sistema.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Eliminar¨ asociado a una pregunta de la lista de chequeo, y después se confirma la eliminación de la pregunta en una ventana en forma de *model*.	
Resultados esperados: La herramienta muestra el listado de preguntas asociadas a la lista de chequeo de uno de los lenguajes en el cual no aparece la pregunta eliminada.	
Evaluación de la prueba: Satisfactoria	

Tabla 84. Prueba de aceptación #33

Caso de prueba de aceptación	
Código: H17_P1	**Historia de usuario:** 17

Nombre: Graficar cantidad de deficiencias por proyecto, por componente revisado
Descripción: Probar la funcionalidad que permite mostrar un gráfico con la cantidad de deficiencias por proyecto, por componente revisado.
Condiciones de ejecución: Debe estar autenticado en el sistema
Entrada/Pasos de ejecución: Se presiona el botón ¨Componentes revisados por proyecto¨
Resultados esperados: La herramienta muestra un gráfico con la cantidad de deficiencias por proyecto, por componente revisado.
Evaluación de la prueba: Satisfactoria

Tabla 85. Prueba de aceptación #34

Caso de prueba de aceptación	
Código: H18_P1	**Historia de usuario:** 18
Nombre: Graficar cantidad promedio de deficiencias por componente, por tipo de componente	
Descripción: Probar la funcionalidad que permite mostrar un gráfico con la cantidad promedio de deficiencias por componente, por tipo de componente.	
Condiciones de ejecución: Debe estar autenticado en el sistema	
Entrada/Pasos de ejecución: Se presiona el botón ¨Tipo de componente ¨	
Resultados esperados: La herramienta muestra un gráfico con la cantidad promedio de deficiencias por componente, por tipo de componente.	
Evaluación de la prueba: Satisfactoria	

Tabla 86. Prueba de aceptación #35

Caso de prueba de aceptación	
Código: H19_P1	**Historia de usuario:** 19
Nombre: Graficar cantidad promedio de deficiencias por lenguaje por tipo de deficiencia	
Descripción: Probar la funcionalidad que permite mostrar un gráfico con la cantidad promedio de deficiencias por lenguaje por tipo de deficiencia.	
Condiciones de ejecución: Debe estar autenticado en el sistema	
Entrada/Pasos de ejecución: Se presiona el botón ¨Lenguaje por tipo de deficiencia ¨	
Resultados esperados: La herramienta muestra un gráfico con la cantidad promedio de deficiencias por lenguaje por tipo de deficiencia.	
Evaluación de la prueba: Satisfactoria	

Tabla 87. Prueba de aceptación #36

Caso de prueba de aceptación	
Código: H20_P1	**Historia de usuario:** 20
Nombre: Graficar cantidad promedio de deficiencias por lenguaje por componente	
Descripción: Probar la funcionalidad que permite mostrar un gráfico con la cantidad promedio de deficiencias por lenguaje por componente.	
Condiciones de ejecución: Debe estar autenticado en el sistema	
Entrada/Pasos de ejecución: Se presiona el botón ¨Deficiencia por lenguaje¨	
Resultados esperados: La herramienta muestra un gráfico con la cantidad promedio de deficiencias por lenguaje por componente.	
Evaluación de la prueba: Satisfactoria	

Tabla 88. Prueba de aceptación #37

Caso de prueba de aceptación	
Código: H21_P1	**Historia de usuario:** 21
Nombre: Generar tabla resumen de RTF	
Descripción: Probar la funcionalidad que permite generar una tabla resumen de RTF	
Condiciones de ejecución: Debe estar autenticado en el sistema y en la vista que muestra el listado de componentes a revisar.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Concluir RTF ¨	
Resultados esperados: La herramienta muestra la tabla resumen	
Evaluación de la prueba: Satisfactoria	

Tabla 89. Prueba de aceptación #38

Caso de prueba de aceptación	
Código: H22_P1	**Historia de usuario:** 22
Nombre: Graficar resultados de RTF	
Descripción: Probar la funcionalidad que permite mostrar un gráfico con los resultados de RTF.	
Condiciones de ejecución: Debe estar autenticado en el sistema y en la vista que muestra el listado de las RTF asociadas a un proyecto.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Mostrar¨	
Resultados esperados: La herramienta muestra un gráfico con los resultados de RTF.	

Evaluación de la prueba: Satisfactoria

Tabla 90. Prueba de aceptación #39

Caso de prueba de aceptación	
Código: H23_P1	**Historia de usuario:** 23
Nombre: Generar Informe de RTF	
Descripción: Probar la funcionalidad que permite generar un informe en PDF de la RTF	
Condiciones de ejecución: Debe estar autenticado en el sistema y en la vista que muestra el listado de las RTF asociadas a un proyecto.	
Entrada/Pasos de ejecución: Se presiona el botón ¨Descargar¨	
Resultados esperados: La herramienta muestra un informe en PDF de la RTF.	
Evaluación de la prueba: Satisfactoria	

Tabla 91. Prueba de aceptación #40

Caso de prueba de aceptación	
Código: H24_P1	**Historia de usuario:** 24
Nombre: Mostrar posibles deficiencias identificadas en archivo de lenguaje PHP.	
Descripción: Probar la funcionalidad que permite mostrar posibles deficiencias identificadas en archivo de lenguaje PHP.	
Condiciones de ejecución: Debe estar autenticado en el sistema y haber iniciado la revisión de un archivo.	
Entrada/Pasos de ejecución: -	
Resultados esperados: La herramienta muestra un listado de posibles deficiencias identificadas en el código fuente del archivo a revisar.	
Evaluación de la prueba: Satisfactoria	

Anexo 4. Tareas de ingeniería

Tabla 92. Tarea de ingeniería #2

Tarea de Ingeniería	
Número de la tarea: 2	**Número de la HU:** 2
Nombre de la tarea: Implementar la funcionalidad Gestionar archivo.	
Tipo de tarea: Desarrollo	

Fecha inicio: 5/3/2015	**Fecha fin:** 8/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un archivo en la base de datos del sistema.	

Tabla 93. Tarea de ingeniería #3

Tarea de Ingeniería	
Número de la tarea: 3	**Número de la HU:** 3
Nombre de la tarea: Implementar la funcionalidad gestionar documento.	
Tipo de tarea: Desarrollo	
Fecha inicio: 8/3/2015	**Fecha fin:** 11/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un documento en la base de datos del sistema.	

Tabla 94. Tarea de ingeniería #4

Tarea de Ingeniería	
Número de la tarea: 4	**Número de la HU:** 4
Nombre de la tarea: Implementar la funcionalidad gestionar RTF.	
Tipo de tarea: Desarrollo	
Fecha inicio: 11/3/2015	**Fecha fin:** 14/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver una RTF en la base de datos del sistema.	

Tabla 95. Tarea de ingeniería #5

Tarea de Ingeniería	
Número de la tarea: 5	**Número de la HU:** 5
Nombre de la tarea: Implementar la funcionalidad gestionar deficiencia de archivo.	
Tipo de tarea: Desarrollo	
Fecha inicio: 14/3/2015	**Fecha fin:** 17/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	

Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver una deficiencia de archivo en la base de datos del sistema.

Tabla 96. Tarea de ingeniería #6

Tarea de Ingeniería	
Número de la tarea: 6	**Número de la HU:** 6
Nombre de la tarea: Implementar la funcionalidad gestionar deficiencia de documento.	
Tipo de tarea: Desarrollo	
Fecha inicio: 17/3/2015	**Fecha fin:** 20/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver una deficiencia de documento en la base de datos del sistema.	

Tabla 97. Tarea de ingeniería #8

Tarea de Ingeniería	
Número de la tarea: 8	**Número de la HU:** 8
Nombre de la tarea: Implementar la funcionalidad gestionar impacto.	
Tipo de tarea: Desarrollo	
Fecha inicio: 27/3/2015	**Fecha fin:** 30/3/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un impacto en la base de datos del sistema.	

Tabla 98. Tarea de ingeniería #9

Tarea de Ingeniería	
Número de la tarea: 9	**Número de la HU:** 9
Nombre de la tarea: Implementar la funcionalidad gestionar tipo de proyecto.	
Tipo de tarea: Desarrollo	
Fecha inicio: 30/3/2015	**Fecha fin:** 3/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un tipo de proyecto en la base de datos del sistema.	

Tabla 99. Tarea de ingeniería #10

Tarea de Ingeniería	
Número de la tarea: 10	**Número de la HU:** 10
Nombre de la tarea: Implementar la funcionalidad gestionar tipo de documento.	
Tipo de tarea: Desarrollo	
Fecha inicio: 3/4/2015	**Fecha fin:** 6/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un tipo de documento en la base de datos del sistema.	

Tabla 100. Tarea de ingeniería #11

Tarea de Ingeniería	
Número de la tarea: 11	**Número de la HU:** 11
Nombre de la tarea: Implementar la funcionalidad gestionar lenguaje de programación.	
Tipo de tarea: Desarrollo	
Fecha inicio: 6/4/2015	**Fecha fin:** 9/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un lenguaje de programación en la base de datos del sistema.	

Tabla 101. Tarea de ingeniería #12

Tarea de Ingeniería	
Número de la tarea: 12	Número de la HU: 12
Nombre de la tarea: Implementar la funcionalidad gestionar tipo de deficiencia de archivo.	
Tipo de tarea: Desarrollo	
Fecha inicio: 9/4/2015	**Fecha fin:** 12/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un tipo de deficiencia de archivo en la base de datos del sistema.	

Tabla 102. Tarea de ingeniería #13

Tarea de Ingeniería

Número de la tarea: 13	**Número de la HU:** 13
Nombre de la tarea: Implementar la funcionalidad gestionar tipo de deficiencia de documento.	
Tipo de tarea: Desarrollo	
Fecha inicio: 12/4/2015	**Fecha fin:** 15/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver un tipo de deficiencia de documento en la base de datos del sistema.	

Tabla 103. Tarea de ingeniería #14

Tarea de Ingeniería	
Número de la tarea: 14	**Número de la HU:** 14
Nombre de la tarea: Implementar la funcionalidad gestionar complejidad de deficiencia.	
Tipo de tarea: Desarrollo	
Fecha inicio: 15/4/2015	**Fecha fin:** 18/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver una complejidad de deficiencia de archivo en la base de datos del sistema.	

Tabla 104. Tarea de ingeniería #15

Tarea de Ingeniería	
Número de la tarea: 15	**Número de la HU:** 15
Nombre de la tarea: Implementar la funcionalidad gestionar lista de chequeo de documento.	
Tipo de tarea: Desarrollo	
Fecha inicio: 18/4/2015	**Fecha fin:** 21/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver una lista de chequeo de documento en la base de datos del sistema.	

Tabla 105. Tarea de ingeniería #16

Tarea de Ingeniería	
Número de la tarea: 16	**Número de la HU:** 16
Nombre de la tarea: Implementar la funcionalidad gestionar lista de chequeo de archivo.	

Tipo de tarea: Desarrollo	
Fecha inicio: 21/4/2015	**Fecha fin:** 24/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite adicionar, eliminar, editar y ver una lista de chequeo de archivo en la base de datos del sistema.	

Tabla 106. Tarea de ingeniería #18

Tarea de Ingeniería	
Número de la tarea: 18	**Número de la HU:** 18
Nombre de la tarea: Implementar la funcionalidad graficar cantidad de deficiencias por tipo de componente.	
Tipo de tarea: Desarrollo	
Fecha inicio: 27/4/2015	**Fecha fin:** 29/4/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón	
Descripción: Se implementa la funcionalidad que permite mostrar una gráfica que muestra la cantidad de deficiencias por tipo de componente.	

Tabla 107. Tarea de ingeniería #19

Tarea de Ingeniería	
Número de la tarea: 19	**Número de la HU:** 19
Nombre de la tarea: Implementar la funcionalidad graficar cantidad de deficiencias por lenguaje por tipo de deficiencia.	
Tipo de tarea: Desarrollo	
Fecha inicio: 29/4/2015	**Fecha fin:** 2/5/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite mostrar una gráfica que muestra la cantidad de deficiencias por lenguaje por tipo de deficiencia.	

Tabla 108. Tarea de ingeniería #20

Tarea de Ingeniería	
Número de la tarea: 20	**Número de la HU:** 20

Nombre de la tarea: Implementar la funcionalidad graficar cantidad promedio de deficiencias por lenguaje por componente.	
Tipo de tarea: Desarrollo	
Fecha inicio: 2/5/2015	**Fecha fin:** 4/5/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite mostrar una gráfica que muestra la cantidad promedio de deficiencias por lenguaje por componente.	

Tabla 109. Tarea de ingeniería #21

Tarea de Ingeniería	
Número de la tarea: 21	**Número de la HU:** 21
Nombre de la tarea: Implementar la funcionalidad generar tabla resumen de RTF.	
Tipo de tarea: Desarrollo	
Fecha inicio: 4/5/2015	**Fecha fin:** 7/5/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite generar la tabla resumen de RTF.	

Tabla 110. Tarea de ingeniería #22

Tarea de Ingeniería	
Número de la tarea: 22	**Número de la HU:** 22
Nombre de la tarea: Implementar la funcionalidad graficar resultado de RTF.	
Tipo de tarea: Desarrollo	
Fecha inicio: 7/5/2015	**Fecha fin:** 9/5/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite graficar el resultado de las RTF.	

Tabla 111. Tarea de ingeniería #23

Tarea de Ingeniería	
Número de la tarea: 23	**Número de la HU:** 23
Nombre de la tarea: Implementar la funcionalidad generar Informe de RTF.	
Tipo de tarea: Desarrollo	

Fecha inicio: 9/5/2015	**Fecha fin:** 12/5/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite generar un informe PDF de la RTF.	

Tabla 112. Tarea de ingeniería #24

Tarea de Ingeniería	
Número de la tarea: 24	**Número de la HU:** 24
Nombre de la tarea: Implementar la funcionalidad mostrar posibles deficiencias identificadas en archivo de lenguaje PHP.	
Tipo de tarea: Desarrollo	
Fecha inicio: 12/5/2015	**Fecha fin:** 31/5/2015
Programador responsable: Eddy Rubén Rodríguez y Daynier Guerrero Chacón.	
Descripción: Se implementa la funcionalidad que permite mostrar posibles deficiencias identificadas en un archivo de lenguaje PHP.	

Anexo 5. Listas de Chequeo

Lista de chequeo para el lenguaje PHP

1. Añade un solo espacio después de cada delimitador coma.
2. Añade un solo espacio alrededor de los operadores (==, &&, ...).
3. Añade una coma después de cada elemento del arreglo en un arreglo multilínea, incluso después del último.
4. Añade una línea en blanco antes de las declaraciones *return*, a menos que el valor devuelto solo sea dentro de un grupo de declaraciones (tal como una declaración *if*).
5. Usa llaves para indicar la estructura del cuerpo de control, independientemente del número de declaraciones que contenga.
6. Declara las propiedades de clase antes que los métodos.
7. Declara los métodos públicos (*public*) primero, luego los protegidos (*protected*), y finalmente los privados (*private*).
8. Los mensajes de excepción se deben concatenar utilizando la función *sprintf*.

9. Utiliza mayúsculas intercaladas —sin guiones bajos— en nombres de variable, función, método o argumentos.
10. Usa guiones bajos para nombres de opción y nombres de parámetro.
11. Utiliza espacios de nombres para todas las clases.
12. Prefija las clases abstractas con *Abstract*. Por favor, ten en cuenta que algunas de las primeras clases de Symfony2 no siguen esta convención y no se han
13. rebautizado por razones de compatibilidad hacia atrás.
14. Sufija las interfaces con Interface.
15. Sufija las características con *Trait*.
16. Sufija las excepciones con *Exception*.
17. Utiliza caracteres alfanuméricos y guiones bajos para los nombres de archivo.
18. El mensaje de la depreciación debería indicar la versión cuándo el método/clase fue depreciado, la versión cuándo será removido, y siempre que sea posible, cómo sustituir la característica.
19. El código escrito en PHP 5.3 o superior debe usar espacios de nombres formales.
20. El código escrito en PHP 5.2.x y anterior, podría usar la convención de pseudo-espacios de nombres con el prefijo _vendor delante de las clases.
21. Las constantes deben declararse en mayúscula y con el caracter "_" como separador.
22. Se evade intencionalmente cualquier recomendación del uso de $StudlyCaps, $camelCase, or $under_score para el nombre de los atributos de las clases.
23. Utiliza paréntesis cuando se instancien clases, independientemente del número de argumentos que tenga el constructor.
24. Define una clase por archivo — esto no se aplica a las clases ayudantes privadas, de las cuales no se tiene la intención de crear una instancia desde el exterior y por lo tanto no les preocupa la norma PSR-0.
25. Los métodos se deben declarar en estilo camelCase.

26. Ninguna línea debe exceder los 120 caracteres de longitud.
27. Debe haber una línea en blanco después de los espacios de nombres y debajo de los bloques “use”.
28. Las llaves de las clases se deben abrir en la línea siguiente a donde se comenzó a declarar esta, y cerrarse en una línea después del cuerpo de la clase.
29. Las llaves de los métodos se deben abrir en la línea siguiente a donde se comenzó a declarar este, y cerrarse en una línea después del cuerpo del método.
30. La visibilidad de los atributos de las clases siempre se declara.
31. La visibilidad de los métodos siempre se declara.
32. Se deben declarar *abstract* y final antes de la visibilidad; *static* se debe declarar después de la visibilidad.
33. Las estructuras de control deben tener un espacio detrás, los métodos y las llamadas de estos no deberían tenerlo.
34. Las llaves de las estructuras de control se deben abrir en la misma línea donde se comenzaron a declarar estás, y cerrarse en una línea después del cuerpo de la estructura.
35. Los paréntesis de apertura en las estructuras de control no deben tener un espacio después de estos, y tampoco deben tener un espacio antes del paréntesis de cerrado.
36. La etiqueta ?> debe omitirse en los archivos que contengan solo código PHP.
37. Todos los archivos PHP deben terminar con una simple línea en blanco.
38. Las líneas con más de 120 caracteres deben picarse en otras de 80 caracteres.
39. No debe haber espacios en blanco al final de las líneas no vacías.
40. Las líneas en blanco se deben añadir para mejorar la legibilidad del código y para separar bloques relacionados de códigos.
41. No debe existir más de una instrucción por línea.
42. Se deben utilizar 4 espacios para identar el código, no utilizar tabulaciones.
43. Las palabras clave deben estar en minúscula, al igual que las constantes PHP *true*, *false* y *null*.

44. Debe haber solo una instrucción use por declaración.
45. Las instrucciones *extends* e *implements* deben declararse en la misma línea de donde se declara la clase.
46. La lista de *implements* pudiera estar separada en múltiples líneas, donde cada línea tendrá un indentado (4 espacios). El primer elemento de la
47. lista debe estar en la siguiente línea de la declaración de la clase, y solo debe haber una interfaz por línea.
48. La palabra reservada *var* no debe usarse para declarar los atributos de las clases.
49. No debe haber más de un atributo de clase declarado por instrucción.
50. Los nombres de los atributos de las clases no deberían tener como prefijo el caracter "_" para indicar la visibilidad *protected* o *private*.
51. Los nombres de los métodos no deberían tener como prefijo el caracter "_" para indicar la visibilidad *protected* o *private*.
52. Los nombres de los métodos no deben tener un espacio atrás.
53. En la declaración de los métodos no debe haber un espacio después del paréntesis ni antes de cerrarlo.
54. En los parámetros de los métodos no debe haber un espacio antes de la coma, y si después de esta.
55. Los argumentos con valores por defecto deben estar al final de la lista de argumentos.
56. Los argumentos pudiera estar separada en múltiples líneas, donde cada línea tendrá un indentado (4 espacios). El primer argumento de la
57. lista debe estar en la siguiente línea de la declaración de la clase, y solo debe haber un argumento por línea.
58. Cuando la lista de argumentos se separe en múltiples líneas, el paréntesis que cierra y la llave que abre deben estar en la línea debajo del último argumento y con un espacio entre los dos.
59. Debe haber un espacio después de la palabra clave de la estructura de control.

60. Debe haber un espacio entre el paréntesis que cierra y la llave que abre.
61. El cuerpo de la estructura de control debe tener un indentado.
62. La llave que cierra debe estar una línea debajo del cuerpo.
63. Una estructura *if* debe lucir como en la imagen.
64. Una estructura *switch* debe lucir como en la imagen.
65. Una estructura *while* debe lucir como en la imagen.
66. Una estructura do *while* debe lucir como en la imagen.
67. Una estructura *for* debe lucir como en la imagen.
68. Una estructura *foreach* debe lucir como en la imagen.
69. Un bloque try catch debe lucir como en la imagen.
70. Las clausuras o funciones anónimas deben declararse con un espacio después de la palabra *function* y un espacio antes y después de la palabra *use*.
71. En las clausuras o funciones anónimas la llave que abre debe estar en la misma línea y la llave que cierra debe ir en la línea después del cuerpo.
72. En las clausuras o funciones anónimas debe haber espacio después del paréntesis que abre la lista de argumentos o de variables, y no debe haber un espacio antes del paréntesis que cierra.
73. En la lista de argumentos y variables de las clausuras o funciones anónimas, no debe haber espacio delante de cada coma, pero debe haber un espacio después de cada una.""

Lista de chequeo de la Guía de instalación y configuración

1. La portada del documento debe contener el logo correspondiente, nombre de la aplicación, versión de la aplicación, fecha e institución a la cual pertenece.
2. El Control del documento debe tener un título, fecha de publicación e identificador. Además una tabla donde se muestran los roles con acceso al documento.

3. La sección Reglas de confidencialidad debe contener una clasificación, una forma de distribución y un texto donde se exponen los elementos más relevantes sobre la confidencialidad del producto.
4. El documento debe contener una tabla de Control de cambios.
5. El prólogo del documento debe contener una Audiencia, un Acceso a Documentación, un Documentos Relacionados, una Sintaxis de Comandos y unas Convenciones Tipográficas.
6. La primera sección del documento debe contar con una breve descripción del negocio automatizado, las mejoras introducidas en los procesos y el proceso de la aplicación. Se pueden hacer además referencia a aspectos generales de la solución propuesta. Verifique la ortografía, coherencia, claridad de las ideas de la sección.
7. El documento debe contener una segunda sección llamada Artefactos del análisis. En la misma deben aparecer los Requisitos Funcionales, los Requisitos No Funcionales, Diagramas de Casos de Uso del Sistema, y la Descripción de los Casos de Uso del Sistema. Verifique la ortografía, coherencia y claridad de las ideas expuestas en la sección.
8. En la subsección Diagrama de Casos de Uso del Sistema se debe mostrar la figura con el subtítulo debajo de esta, de forma centrada, en negritas, con 6 puntos de interlineado por encima y 10 puntos por debajo.
9. En la sección Artefactos del diseño se deben presentar los diagramas de clases del diseño.
10. Debe verificar la aplicación de las convenciones tipográficas definidas en el documento a lo largo de este.
11. La sección Arquitectura del Sistema debe contar con dos subsecciones, la primera Descripción de la Arquitectura del Sistema y la segunda Diagrama de despliegue.
12. En la subsección Descripción de la Arquitectura del Sistema deberá aparecer una breve descripción de los elementos fundamentales de la arquitectura que fueron considerados para la implementación de la solución.
13. En la subsección Diagrama de Despliegue deberá aparecer el diagrama de despliegue del sistema en una figura según las convenciones tipográficas definidas.

14. En la sección de Base de Datos debe contar con tres subsecciones la primera Diagrama Entidad Relación de la base de datos, la segunda Descripción de las entidades de la base de datos y la última Diccionario de Datos.
15. En el epígrafe Diagrama Entidad Relación de la base de datos deberá aparecer el diagrama entidad relación en una figura según las convenciones tipográficas pautadas.
16. En el epígrafe Descripción de las entidades de la base de datos deberá aparecer una breve descripción de las entidades de la base de datos.
17. En el epígrafe Diccionario de Datos deberán aparecer listados todos los términos incluidos en el diccionario de datos de la aplicación.
18. La sección Novedades de la Versión debe contar con un epígrafe llamado Nuevas Funcionalidades en el que se listan todas las nuevas funcionalidades del sistema así como una breve descripción de la misma.
19. La sección Vista General de Instalación debe describir los pasos necesarios para lograr una instalación exitosa de la aplicación informática.
20. La sección Vista General de la Instalación debe contar con los epígrafes Planificación de la instalación, Instalación y Configuración del Sistema Operativo, Instalación y Configuración de la Base de Datos, Instalación y Configuración del Servidor de Aplicaciones, Instalación de Aplicaciones Adicionales, Instalación de Actualizaciones y Pruebas de Instalación.
21. El epígrafe Planificación de la Instalación debe presentar las siguientes fases: Leer las notas de liberación, Plan de Instalación, Tareas de Pre-instalación, Instalación, Tareas de Pos-instalación y Configuración de <Aplicación informática>
22. El epígrafe Instalación y Configuración del Sistema Operativo debe brindar información necesaria para realizar la instalación de <Sistema Operativo>. Debe contener además los pasos para la instalación del sistema operativo.
23. El epígrafe Instalación y Configuración de la Base de Datos debe brindar información necesaria para realizar la instalación de <Base de datos> con los pasos incluidos.
24. El epígrafe Instalación y Configuración del Servidor de Aplicaciones debe brindar información necesaria para realizar la instalación de <Servidor de Aplicaciones> con los pasos incluidos.

25. El epígrafe Instalación de Aplicaciones Adicionales debe brindar información necesaria para realizar la instalación de <Aplicación> con los pasos incluidos.

26. El epígrafe Instalación de Actualizaciones debe brindar información necesaria para realizar las actualizaciones correspondientes a <Aplicación Informática> con los pasos incluidos.

27. El epígrafe Pruebas de la Instalación debe brindar información necesaria para realizar pruebas a la instalación de <Servidos de Aplicaciones> con los pasos incluidos.

28. La sección Requerimientos de Instalación de <Aplicación Informática> debe proporcionar información sobre las precondiciones para la instalación de la aplicación.

29. La sección Requerimientos de Instalación de <Aplicación Informática> debe contener las siguientes secciones: Asignación de permisos, Requerimientos de hardware, Requerimientos de *Software*, Comprobación de la configuración de red, Instalación de paquetes para Linux, Creación de los grupos de usuarios requeridos por el sistema operativo, Verificación de los límites de recursos para el usuario de instalación de <Aplicación Informática>, Creación e identificación de los directorios requeridos para la instalación del *software*, Identificación o creación de un directorio base, Selección de una opción de almacenamiento para los archivos de recuperación, Creación de directorios para los archivos de recuperación, Configuración de los dispositivos de disco para <Aplicación Informática>, Detención de procesos que interfieran en la instalación.

30. La sección Instalación de <Aplicación Informática> debe comenzar con una breve descripción del instalador.

31. La sección Instalación de <Aplicación Informática> debe contener los siguientes epígrafes: Consideraciones previas a la instalación, Acceso a la instalación del *software* (vías de acceso al instalador), Opciones de seguridad (políticas de expiración de contraseñas, auditorias del sistema), Instalación del *software* (si hay interfaz gráfica debe haber una tabla de dos columnas: Pantalla y Acción, donde se listen todas las pantallas y acciones posibles) y por último Intalación de ejemplos (debe referenciar a tutoriales o materiales que contengan ejemplos del uso del instalador).

32. La sección Tareas de Pos-instalación de <Aplicación Informática> debe describir cómo completar las tareas posteriores a la instalación del *software*.

33. La sección Tareas de Postinstalación de <Aplicación Informática> debe incluir además información sobre los siguientes temas: Tareas requeridas después de la instalación (como instalación de parches) y Tareas recomendadas después de la instalación (como habilitar y deshabilitar opciones).

34. La sección Configuración de <Aplicación Informática> debe ofrecer información sobre la configuración por defecto de la aplicación, incluyendo datos sobre las cuentas, contraseñas y ubicaciones de archivos.

35. La sección Configuración de <Aplicación Informática> también debe contar con los necesarios de los siguientes tópicos: Verificar el contenido de la instalación de <Aplicación informática> y la ubicación del directorio, Inicio de sesión en <Aplicación Informática>, Revisión de cuentas y contraseñas, Desbloqueo y Restauración de contraseñas de usuario, Localizar el archivo de parámetros del servidos, Revisión de tablas y archivos de datos, rehacer los archivos de registro y archivos de control.

36. La sección Desinstalación de <Aplicación Informática> debe describir como desinstalar el *software* completamente del sistema y la configuración de los archivos relacionados con la instalación.

37. La sección Preguntas Frecuentes debe incluir las preguntas frecuentes más significativas relacionadas con la instalación de la aplicación.

38. El documento debe incluir un glosario de términos donde se deben aclarar todos aquellos términos que puedan ser objeto de duda para el lector.

Lista de chequeo para el Manual de usuario

1. La portada del Manual de usuario debe mencionar correctamente el nombre de la aplicación con su versión al cual pertenece. Además debe tener la fecha de elaboración y el logo de la institución.

2. El índice de contenidos debe estar integrado fundamentalmente por los siguientes elementos: prólogo, Configuración inicial, Seguridad y control de acceso, Subsistemas o módulos, Preguntas frecuentes y Glosario de términos.

3. El prólogo debe contener las siguientes secciones: Audiencia, Acceso a documentación, Documentos relacionados y Convenciones tipográficas.

4. La sección de Audiencia debe especificar a que tipo de usuario está dirigido el documento, por ejemplo administradores de red, ingenieros en informática, entre otros.
5. La sección Acceso a documentación debe describir las vías de acceso, políticas de uso y formatos de la documentación relacionada con la aplicación informática.
6. La sección Documentos relacionados debe hacer referencia a otros documentos de utilidad para el usuario o que se relacionen con el manual en cuestión.
7. La sección Convenciones tipográficas especifica las convenciones tipográficas a seguir en el documento. Debe contener una tabla donde se haga referencia a cada una de estas convenciones.
8. Las imágenes que se utilicen de guía en el manual deben incluir descripciones breves claras y precisas.
9. Las imágenes y figuras incluidos gráficos deben ser colocadas cercanas al texto que las menciona, deben ser además enumeradas con números arábicos en orden ascendente a medida que se empleen.
10. Los subtítulos de las figuras deben estar centrados y debajo de esta, en negritas con 6 puntos de interlineado por encima y 10 puntos por debajo.
11. Las imágenes utilizadas no se deben dividir, verse claramente y ser legibles.
12. El capítulo de Configuración inicial se debe incluir en los manuales de aplicaciones donde el usuario debe realizar algún tipo de configuración. Los textos se deben apoyar con imágenes que ayuden al entendimiento por parte del usuario.
13. El capítulo Seguridad y control de acceso debe describir las formas de autenticación de la herramienta, niveles de acceso del usuario al que está dirigido el manual y demás cuestiones de seguridad relacionadas con el sistema. Los textos se deben apoyar en imágenes siguiendo las convenciones tipográficas antes establecidas.
14. El capítulo subsistemas o módulos debe detallar las funcionalidades del sistema objeto del documento. Se pueden añadir tantas secciones como sean necesarias para describir todas las funcionalidades del sistema, preferiblemente una por módulo.

15. Las explicaciones de las diferentes funcionalidades de la aplicación descrita en el manual deben realizarse en forma de pasos lógicos y ser apoyadas con imágenes de las pantallas que verá el usuario.
16. La sección de Preguntas frecuentes debe incluir aquellas preguntas frecuentes en los usuarios hacia los cuales va dirigido el manual y sus respuestas.
17. La sección Glosario de términos debe aclarar todos aquellos términos que puedan ser objeto de duda para el lector.

Anexo 6. Entrevista

1. ¿Siempre que usted ejerce como programador en el proyecto al cual pertenece conoce con claridad el estilo de código que debe utilizar?¿En caso de conocerlo, alguien se encarga en el proyecto en la medida que usted programa de verificar que lo está cumpliendo?
2. ¿Qué lenguajes de programación utiliza frecuentemente?
3. ¿Cree usted que el centro al cual pertenece o la propia universidad debieran definir estándares de codificación propios?
4. ¿Ha tenido usted que enfrentarse a la situación de programar parte de código fuente ya comenzado y no acabado por otra persona?¿En caso de haberlo hecho qué dificultades ha tenido que enfrentar?
5. ¿Las validaciones de datos de los formularios como las realiza cuando programa?
6. ¿Cree usted que los estilos de código influyen en el rendimiento de una aplicación?
7. ¿Ha tenido usted que trabajar en el mantenimiento de alguna aplicación?¿En caso de haberlo hecho qué dificultades ha tenido que enfrentar?
8. ¿Participa usted de alguna forma en la elaboración de los documentos Guía de instalación y configuración y Manual de usuario?
9. ¿Cómo programador qué entregables recibe antes de comenzar la etapa de Implementación?