Universidad de las Ciencias Informáticas

Facultad 3

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

Título: Herramienta para la estimación de tiempo y esfuerzo de las pruebas de aceptación, liberación y piloto.

Autor: Osmail Blanco Pérez

Tutores: Ing. Mairelys Martínez López

Ing. Doris Maza Oval

La Habana, junio de 2014

"Año 56 de la Revolución"

...Un hombre de estos tiempos nutrido exclusivamente de conocimientos literarios es como un mendigo flaco y hambriento cubierto con un manto esmaltado de joyas de riquísima púrpura…

José Martí.

DECLARACIÓN DE AUTORÍA

Mediante estas palabras queda declarado que soy autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Osmail Blanco Pérez.

_____

Firma del Autor

Ing. Mairelys Martínez López.

_____

Firma del Tutor

Ing. Doris Maza Oval.

_____

Firma del Tutor

DATOS DE CONTACTO

Autor:

Osmail Blanco Pérez

E-mail: oblanco@estudiantes.uci.cu

Tutores:

Ing. Mairelys Martínez López

E-mail: mmlopez@uci.cu

Ing. Doris Maza Oval

E-mail: dmaza@uci.cu

AGRADECIMIENTOS

A mis padres y abuelos que con tanta abnegación y sacrificio han contribuido al logro de este, mi gran anhelo.

A mis tutoras Mairelys Martínez López y Doris Maza Oval por su valiosa ayuda para la realización de este trabajo.

A mis profesores que durante cinco años me han conducido por los saberes de las nuevas tecnologías de la información y las comunicaciones.

A José y Francisco que siempre me han acompañado en los momentos más difíciles de mi vida y mis estudios.

A todas las personas que han compartido alegrías y pesares durante mi vida como estudiante universitario.

A todos, muchas gracias.

DEDICATORIA

A mi abuelo Alfonso Blanco que en vida desbordó amor y ternura cual su propio hijo acompañado por alguien para quien yo soy más que eso, mi abuela Aleida.

A mis padres y hermanos que me han permitido gozar de una hermosa familia.

A mi abuela Chavela, tíos y primos.

A José, alguien muy especial para mí.

RESUMEN

En el desarrollo de software, la estimación constituye un factor fundamental para garantizar la utilización correcta de los recursos humanos y materiales de un proyecto. En varios proyectos de desarrollo de software del mundo, los expertos en realizar estimaciones emplean el “Método de estimación de tiempo y esfuerzo” con el objetivo de organizar el trabajo y lograr una mejor planificación.

El objetivo de la presente investigación, es desarrollar una herramienta para estimar el tiempo y el esfuerzo empleado durante la realización de las pruebas de aceptación, liberación y piloto en el grupo de calidad del Centro de Gobierno Electrónico de la Universidad de las Ciencia Informáticas.

Después de un estudio realizado se definieron las herramientas y tecnologías necesarias para el desarrollo de la aplicación y se seleccionó XP como metodología para guiar el proceso de desarrollo del software. Se identificaron mediante historias de usuario las funcionalidades que debe cumplir la herramienta. Además se realizó el diseño y se llevó a cabo la implementación para obtener la solución propuesta. Como resultado, se obtuvo una herramienta de escritorio capaz de realizar estimaciones en el grupo de calidad. Se realizaron pruebas de aceptación, pruebas de caja blanca y caja negra para comprobar que la herramienta funciona correctamente.

PALABRAS CLAVES: esfuerzo, estimación, planificación, pruebas, tiempo.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El proceso de desarrollo de software se ha convertido en una actividad fundamental en el mundo para la informatización de la sociedad. En Cuba este proceso se lleva a cabo para crear soluciones informáticas en las diferentes esferas de la sociedad. Uno de los resultados más relevantes de la Revolución Cubana es sin duda alguna, la formación humana y el desarrollo profesional y científico, esbozada como política por el Comandante en Jefe Fidel Castro, el 15 de enero de 1960, cuando planteara: “El futuro de la Patria tiene que ser, necesariamente, un futuro de hombres de ciencia”.

El desarrollo del modelo cubano para el perfeccionamiento de aplicaciones informáticas en la industria nacional es el principal reto hacia el cual se proyectan las acciones del Centro Nacional de Calidad de Software (Calisoft). En esta tarea juega un papel importante la Universidad de las Ciencias Informáticas (UCI), que cuenta con un conjunto de centros de desarrollo de software como es el caso del Centro de Gobierno Electrónico (CEGEL).

CEGEL actualmente cuenta con un grupo de calidad en el cual se le realizan un conjunto de pruebas al software durante su proceso de desarrollo, con la finalidad de que el producto cumpla con las expectativas y requisitos del cliente.

Las pruebas de software son un conjunto de validaciones que se realizan a todo software, para avalarlo como apto para entregarse al cliente. Dentro de estas pruebas ocupan gran importancia:

- Las pruebas de aceptación, que tienen como objetivo validar que un sistema cumpla con el funcionamiento esperado.
- Las pruebas de liberación, las cuales se realizan en la etapa final del desarrollo del software y previo a las pruebas de aceptación y despliegue.
- Las pruebas piloto que son la forma de probar una aplicación determinada.

Para garantizar la calidad de estas pruebas se requiere de cierta cantidad de recursos y de tiempo que le permita a los especialistas del grupo de calidad trabajar de forma

organizada en los diferentes productos a liberar. Para el centro es necesario realizar estimaciones para lograr un mejor uso de la fuerza de trabajo disponible en el grupo de calidad y lograr una mejor planificación de las pruebas de software.

La estimación es una predicción aproximada que tiene una probabilidad de ser cierta. En el orden internacional es poco usada en la fase de pruebas, siendo imprescindible debido a los constantes cambios en el desarrollo de software. En la fase de pruebas de software, más específicamente en las pruebas de aceptación, liberación y piloto, es necesaria la aplicación de un método de estimación de tiempo y esfuerzo, para lograr así tener estimaciones lo más cercana a la realidad, que se traducirán en una mejor planificación de los recursos humanos y materiales.

En CEGEL durante las pruebas de liberación, aceptación y piloto, la planificación de los recursos materiales y humanos que participarán en el proceso de pruebas se realiza a partir de la experiencia personal del equipo de trabajo, lo que provoca que no se realicen estimaciones referentes al uso adecuado de los recursos y al tiempo establecido para ejecutarlas, utilizando en algunas ocasiones más tiempo del planificado. Además no se establecen criterios de complejidad de los artefactos a probar por lo que no se sabe cuánto durará el proceso. Tampoco se identifican los riesgos que pueden afectar el desarrollo normal de las pruebas por lo que no se tienen en cuenta un grupo de factores que influyen directamente en la estimación del esfuerzo que se aplica, entre los factores se encuentran: entrada sin retrasos de la documentación al proceso de pruebas y cumplimiento de lo planificado en el cronograma de prueba por parte del equipo de desarrollo, entre otros.

Por todo lo antes expuesto surge el siguiente **problema a resolver**: ¿Cómo contribuir a la mejora del proceso de estimación de tiempo y esfuerzo de las pruebas de liberación, aceptación y piloto del Centro de Gobierno Electrónico de manera que facilite la planificación de las pruebas?

Después de analizar el problema planteado se tiene como **objeto de estudio:** Estimación de tiempo y esfuerzo en el proceso de pruebas de software.

Para encontrar una solución se definió como **objetivo general**: Desarrollar una herramienta para la estimación de tiempo y esfuerzo de las pruebas de liberación,

aceptación y piloto en el Centro de Gobierno Electrónico que contribuya a la correcta planificación del proceso de pruebas.

La presente investigación tiene como **campo de acción**: Estimación de tiempo y esfuerzo en el proceso de pruebas de liberación, aceptación y piloto. Analizando lo antes planteado se tiene como **idea a defender**: Con el desarrollo de una herramienta para la estimación de tiempo y esfuerzo de las pruebas de liberación, aceptación y piloto en el centro CEGEL, se contribuye a mejorar la planificación del proceso de pruebas.

Derivándose los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación para fundamentar las bases de la solución.
- Desarrollar la propuesta de solución.
- Validar la propuesta de solución.

Para desarrollar el presente trabajo se definieron las siguientes **tareas investigativas**:

- Estudio del estado del arte de los diferentes conceptos relacionados con el tema, así como de las herramientas existentes para la estimación.
- Definición de la metodología, lenguaje y herramientas para darle solución al problema planteado.
- Modelación del negocio para describir la propuesta de solución de manera clara que facilite la captura de los requisitos.
- Captura de requisitos para obtener las especificaciones funcionales y no funcionales que debe tener la herramienta.
- Elaboración del análisis y diseño de la herramienta para obtener la modelación necesaria como base para la implementación de los requisitos definidos.
- Implementación de los requisitos definidos para darle solución al problema planteado.
- Realización de pruebas de software para demostrar que la propuesta de solución resuelve el problema planteado.

Con el fin de cumplir las tareas investigativas, se hace uso de los siguientes **Métodos científicos**:

Métodos teóricos:

- Histórico – Lógico: Se empleó para encontrar información de carácter histórico sobre los métodos y herramientas existentes en el mundo para cálculo de estimación.
- Modelación: Se utilizó principalmente para modelar las clases del diseño a desarrollar que se utilizarán en la herramienta.
- Analítico – Sintético: La función de este método, contribuyó para obtener los datos más relevantes relacionados con el objeto de estudio, donde se analizaron diferentes documentos y teorías relacionada con la investigación en curso.

Métodos empíricos:

- Entrevista: Este método fue utilizado con el objetivo de obtener información importante que facilite el desarrollo del presente trabajo, por ejemplo, la obtención de los requisitos de la herramienta.

Para una mayor organización del trabajo se decidió estructurar esta investigación de la siguiente forma:

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA: En este capítulo se conceptualizan los términos importantes para la investigación, se realiza un estudio de los métodos y herramientas de estimación existentes en el mundo y en Cuba, además de que se define la metodología de desarrollo de software, lenguaje de programación y herramientas a utilizar para darle solución al problema planteado.

CAPÍTULO II. DESCRIPCIÓN DE LA PROPUESTA DE SOLUCIÓN: En este capítulo se realiza una descripción de la herramienta propuesta y sus principales funcionalidades. Se detallan los principales artefactos generados en el proceso de desarrollo de la solución.

CAPÍTULO III. VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN: Este capítulo tiene como objetivo validar que la herramienta implementada responde a las necesidades

que se plantearon al inicio de la investigación, para ello se realizan un conjunto de pruebas y se analizan los resultados obtenidos.

CAPÍTULO I: Fundamentación Teórica

1.1- Introducción

A lo largo de éste capítulo se abordan los conceptos fundamentales para la presente investigación. Además se realiza un estudio de los diferentes métodos y herramientas de estimación existentes en el mundo y en Cuba, así como de la metodología de desarrollos de software, herramientas y lenguajes de programación a utilizar.

1.2- Estimación de tiempo y esfuerzo

Según Pressman "la estimación, como actividad antecesora y constante de lo que luego será la planificación, proporciona valores aproximados de costos, tiempos y esfuerzos que se necesitarán para el desarrollo del producto a construir. Esa aproximación, requiere experiencia, acceder a una buena información histórica y el coraje de confiar en predicciones (medidas)" (1).

En otras palabras, es estimar cuánto esfuerzo, costo, recursos y tiempo supondrá construir un sistema o producto de software específico. Estas se realizan al inicio del proyecto antes de que se empiece a desarrollar el producto, pero en la actualidad es poco usada en la fase de prueba de software siendo indispensable que se realice debido a los constantes cambios en el proceso del desarrollo de software.

1.3- Pruebas de software

Las pruebas de software son un conjunto de pruebas que se realizan a todo software, para avalarlo como apto para entregarse al cliente. El sistema no es el único artefacto a probar, su documentación también forma parte del campo de acción de las pruebas, principalmente aquellos documentos que son entregados junto al producto para guiar y ayudar al cliente a interactuar con la herramienta (1).

- **Pruebas de liberación**

El proceso de pruebas de liberación no es más que un conjunto de pruebas que se realizan en la etapa final del desarrollo del software y previo a las pruebas de aceptación y despliegue. Básicamente se centra en la revisión de la documentación y la aplicación del sistema teniendo en cuenta niveles para los que se emplean varios métodos y técnicas de una manera organizada y eficiente, que conducirán a un mejor funcionamiento del producto (1).

- **Pruebas de aceptación**

El objetivo de las pruebas de aceptación es validar que un sistema cumpla con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento. Las pruebas de aceptación las realiza el usuario del sistema y son preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario (1).

- **Pruebas piloto**

Las pruebas piloto son la forma de probar una aplicación determinada. Se utilizan principalmente para obtener las no conformidades y los cambios que necesita el sistema a medida que se está explotando en manos de usuarios reales. Estas pruebas también ayudan a determinar los recursos que necesita la aplicación, cómo debe desarrollarse el despliegue, cómo debe ser la capacitación de los trabajadores, en fin definir los riesgos que puede traer el piloto y la mejor forma de mitigarlos. (1)

1.4- Método de estimación

El método de estimación que se emplea para desarrollar la herramienta, titulado ´´Método de estimación de tiempo y esfuerzo´´, es resultado de una investigación realizada por los autores Alejandro Santanach, Juan E. Vargas, Lizardo Ramírez, Diana R. Prieto y Mayrín Ramos publicado en la Revista Colombiana de Computación (2). El método de estimación tiene como objetivo reducir las dificultades que puedan presentarse a la hora de realizar las estimaciones de las pruebas de aceptación, liberación y piloto. Este método de tiempo y esfuerzo está basado en la unión de modelos de estimación conocidos como COCOMO y Puntos de Función, o sea, se

emplean las principales fórmulas relacionadas con el tiempo y el esfuerzo del modelo COCOMO las cuales se mezclan con las características de sistema que plantea Puntos de Función, para formular finalmente ecuaciones que facilitan el cálculo de la estimación de tiempo y esfuerzo. A continuación se realiza una breve descripción de los modelos utilizados para conformar el ´´Método de estimación de tiempo y esfuerzo´´.

Modelo COCOMO

El modelo de estimación COCOMO es utilizado por miles de administradores de proyectos de software y se basa en un estudio de cientos de proyectos de software. Este modelo permite determinar el esfuerzo y tiempo de un proyecto de software a partir de los Puntos de Función, lo cual significa una ventaja, dado que en la mayoría de los casos es difícil determinar el número de líneas de código de que constará la herramienta, en especial cuando se tiene poca o ninguna experiencia previa en proyectos de software.

Puntos de Función

El Punto de función es una medida genérica utilizada para englobar en una métrica común, varias características diferentes, con el propósito de generalizar un proceso determinado. Dicho de otras palabras, los puntos de función no son más que una medida del tamaño funcional del software. Sus objetivos son:

- Medir lo que el usuario pide y lo que el usuario recibe.
- Medir independientemente de la tecnología utilizada en la implantación del sistema.
- Proporcionar una métrica del tamaño.
- Proporcionar un medio para la estimación del software (2).

Además el método emplea el Factor de Valor Agregado (FVA), este es un valor que se obtiene después de realizar un análisis de los principales riesgos identificados en las pruebas de aceptación, liberación y piloto donde se establecen un grupo de factores que influyen directamente en las estimaciones del esfuerzo que se aplica al realizar las pruebas a los proyectos.

Para poder realizar las estimaciones mediante este método es necesario conocer algunos datos de entrada dependiendo del tipo de artefacto. El grupo de desarrollo puede solicitar las pruebas a grupos de artefactos que pueden ser:

- Documentación de software.
- Aplicaciones.

A continuación se relaciona una tabla con la complejidad para artefactos de tipo aplicación (Alta, Media o Baja), cada clasificación está acotada por una cuota mínima y máxima de puntos de función:

Artefacto(Complejidad)	Cuota Máxima (Puntos de función)	Cuota Mínima (Puntos de función)
Alta	(∞)	30
Media	30	11
Baja	10	1

Tabla 1. Tiempo de prueba por tipo de complejidad aplicación. (2)

La siguiente tabla muestra la duración (en horas) de las pruebas que se le realizan a un artefacto de tipo documentación según su complejidad:

Artefacto(Complejidad)	Duración(horas)
Alta	4
Media	3
Baja	2

Tabla 2. Complejidad de artefactos de tipo documentación. (2)

En el ´´Método de estimación de tiempo y esfuerzo´´, el tiempo que dura la revisión de un documento es una medida necesaria que no puede dejar de tenerse en cuenta y este varía de acuerdo a la complejidad que presenta, como se muestra en la siguiente tabla:

Documentos(Complejidad)	Duración(horas)
Alta	16
Media	8
Baja	4

Tabla 3. Tiempo de duración de un documento. (2)

1.4.1- Recursos disponibles

Para la ejecución del método de estimación es fundamental conocer cierta cantidad de recursos con que cuenta el grupo de calidad, además es necesario plantear que mientras mayor sea la cantidad de recursos disponibles, menor será el tiempo empleado para hacer las pruebas, estos recursos pueden ser:

- Cantidad de computadoras disponibles con las que cuenta el laboratorio para realizar las pruebas.
- Cantidad de probadores que realizarán las pruebas.

1.4.2- Estimación de esfuerzo y tiempo

A continuación se relacionan las ecuaciones para realizar las estimaciones de tiempo y esfuerzo, tanto para aplicaciones como para documentos.

Estimación del esfuerzo para aplicaciones y documentación

La estimación del esfuerzo para las aplicaciones esta dado en horas-hombres, en esta fórmula se relaciona el esfuerzo total que se aplica al realizar las pruebas en aplicaciones, la estimación del esfuerzo se define como:

Eap= (CantA*TA + CantM * TM + CantB * TB) * FVA.

Conociendo que:

Eap: Esfuerzo estimado para las aplicaciones.

CantA: Cantidad de artefactos de complejidad Alta.

CantM: Cantidad de artefactos de complejidad Media.

CantB: Cantidad de artefactos de complejidad Baja.

TA: Tiempo de duración para artefactos de complejidad Alta

TM: Tiempo de duración para artefactos de complejidad Media.

TB: Tiempo de duración para artefactos de complejidad Baja.

FVA: Factor del valor agregado.

La estimación del esfuerzo para la documentación esta dado en horas-hombres y relaciona el esfuerzo total de aplicar las pruebas a la documentación, se define como:

Ed = (CantDA*TDA + CantDM*TDM + CantDB*TDB) * FVA.

Conociendo que:

Ed: Esfuerzo estimado para la documentación.

CantDA: Cantidad de documentos de complejidad Alta.

CantDM: Cantidad de documentos de complejidad Media.

CantDB: Cantidad de documentos de complejidad Baja.

TDA: Tiempo de duración de las pruebas para los documentos de complejidad Alta.

TDM: Tiempo de duración de las pruebas para los documentos de complejidad Media.

TDB: Tiempo de duración de las pruebas para los documentos de complejidad Baja.

FVA: Es el Factor de Valor agregado.

Estimación del tiempo para aplicaciones y documentación

Para estimar el tiempo relacionado a las aplicaciones en las pruebas de aceptación, liberación y piloto se divide el esfuerzo individual aplicado a cada prueba con los recursos para realizar las pruebas, quedando:

Tap=Eap/R.

Conociendo que:

Tap: Tiempo total de desarrollo estimado en las pruebas para las aplicaciones.

Eap: Esfuerzo aplicado individualmente en cada prueba.

R: Recursos disponibles (cantidad de Pc disponibles o probadores que estén trabajando).

Para estimar el tiempo relacionado con la documentación se usa la métrica que a continuación se muestra:

Td = Ed/R

Conociendo que:

Td: Tiempo real de desarrollo estimado para las pruebas de documentación.

1.4.3- Factor de Valor Agregado (FVA)

El uso del FVA, es de gran importancia para realizar las estimaciones, pues brinda un valor numérico que es utilizado como dato en las ecuaciones para estimar artefactos, el FVA se determina a partir del análisis de los principales riesgos identificados en las pruebas de liberación, aceptación y piloto, estableciéndose un grupo de factores que, basándose en las experiencias de los especialistas de prueba, influyen directamente en la estimación del esfuerzo que se aplica al realizar las pruebas a los proyectos. Para calcular el Factor de Valor Agregado se multiplica el factor obtenido por 0,2; después se suma uno al resultado de la multiplicación y posterior a la suma se divide el resultado por el factor asignado a valoraciones pesimistas, como se muestra a continuación:

FVA = 1+FO*0.2/FP

Conociendo que:

FO: factor obtenido de la sumatoria de Pi (Peso de cada factor) * Ci (Complejidad).

FP: factor obtenido asignando valoraciones pesimistas.

Una vez realizado el análisis del método de estimación utilizado para desarrollar la herramienta, se continuó la investigación con el estudio de las aplicaciones existentes relacionadas con el tema del presente trabajo.

1.5- Análisis de las soluciones existentes

En la actualidad es necesario el uso de las tecnologías de la informática para lograr que el proceso de desarrollo de software se efectúe de forma adecuada y con el menor

tiempo y esfuerzo posible. La combinación de datos históricos y técnicas puede ayudar a una mejor precisión de la estimación, aunque esta no sea una ciencia exacta.

1.5.1- Ámbito internacional

En el ámbito internacional existen varias herramientas dedicadas a la estimación de proyectos, a continuación se relaciona una tabla con las herramientas más significativas:

Nombre de la Herramienta	Plataforma	Empresa Vendedora	Observaciones
Estimacs	Excel	Computer Associates International Inc. www.ca.com/products/stimacs.htm	Permite realizar estimaciones de puntos de función
20s Estimation Calculator	Excel	20smackers www.20smackers.com	Permite calcular esfuerzo y costos en etapas tempranas del desarrollo
20s Reference Estimation	Excel	20smackers www.20smackers.com	Estima en base a proyectos históricos
Costar		Softar systems Inc. www.softarsystems.com	COCOMO

Tabla 4. Tipos de herramientas de estimación.

Después de revisar la tabla anterior se procede a realizar un análisis de las herramientas señaladas donde se plantean la función principal que realizan y porque no satisfacen los problemas del grupo de calidad relacionados principalmente con la planificación, destacando como característica común que son herramientas privativas y que resultan de gran costo para ser adquiridas por la universidad.

Estimacs

La herramienta Estimacs ofrece una serie de estimaciones del costo para grandes proyectos de software. Dicha herramienta es utilizada para hacer proyecciones de presupuesto, como se puede observar, no gestiona ningún dato relacionado con la planificación de pruebas. Sin embargo sirvió como guía de estudio para los desarrolladores ya que emplea Puntos de Función.

Estimation Calculator

La herramienta Estimation Calculator realiza estimaciones basadas principalmente en el esfuerzo y en el costo de un proyecto de software, sin embargo esta herramienta está diseñada para trabajar con datos relacionados con los recursos necesarios y el costo, para comenzar un proyecto de software.

Estimation

Estimation es una herramienta diseñada hace más de una década, por lo que no se tiene en cuenta debido a los grandes cambios que ha presentado el desarrollo del software a través de los años. Estimation realiza estimaciones de esfuerzo, pero realiza el cálculo mediante comparaciones de proyectos antiguos con proyectos recién desarrollados.

Costar

Costar es una herramienta de estimación de costos de software basado en el modelo COCOMO, es muy usada por los directores de proyectos de software para producir estimaciones de la duración de un proyecto, los niveles de dotación de personal y el costo. Además permite hacer intercambios de datos y realizar diferentes tipos de análisis para llegar a un plan de proyecto satisfactorio, además está desarrollada para grandes empresas donde se maneja grandes cantidades de recursos humanos y materiales.

1.5.2- Ámbito nacional

Después de realizar una búsqueda, se obtuvo que en la Universidad de las Ciencias Informáticas se ha investigado sobre el tema e incluso se ha implementado una

herramienta relacionada con la estimación de las pruebas de desarrollo de software. A continuación se muestra un ejemplo:

Herramienta para la estimación de proyecto

Esta herramienta permite estimar proyectos, pero no se puede utilizar ya que establece comparaciones entre las características de un proyecto nuevo con otros ya concluidos. Además para esto se utilizará la técnica de estimación por analogía. Otra característica a señalar es que la herramienta no realiza el cálculo de estimación de tiempo para las pruebas piloto.

Una vez concluido el análisis de las características de las herramientas existentes y observando que son muy costosas en su mayoría y que no satisfacen el problema relacionado con la planificación de las pruebas del grupo de calidad, se evidencia una vez más el porqué de la necesidad de implementar una herramienta para el cálculo de estimación de tiempo y esfuerzo en el grupo de calidad del centro CEGEL.

Al finalizar la presente investigación se pretende entregar al grupo de calidad del centro una herramienta que permita estimar el tiempo y esfuerzo de la pruebas de software. Para ello es necesario hacer un análisis de los tipos de aplicaciones que existen con el objetivo de desarrollar la que mayores ventajas brinde.

1.6- Tipo de aplicación

Para analizar un proyecto, se debe conocer qué tipo de aplicación será desarrollada. Es importante conocer si la aplicación es de tipo escritorio o tipo web. Para entender las diferencias entre un tipo de aplicación y otro es necesario ver su arquitectura, la cual se muestra a continuación:

1.6.1- Aplicación Web

En la ingeniería de software se denomina aplicación web a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una Intranet mediante un navegador, además el alto grado de desarrollo actual permite la actualización y el mantenimiento (vía Internet) de dichas aplicaciones, sin que se deba distribuir e instalar algún software en específico.

Una aplicación web es un programa especialmente diseñado para ejecutarse dentro de un navegador web. La figura que a continuación se muestra constituye un ejemplo de arquitectura para este tipo de aplicaciones:

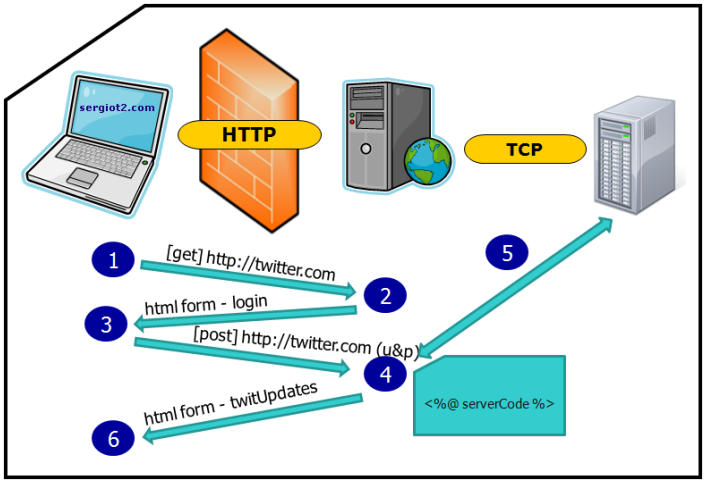

Fig.1. Arquitectura de una aplicación web. (3)

Conociendo que:

1. El usuario ingresa la URL de la página en su navegador. El navegador por detrás hace una solicitud al servidor web usando el protocolo de comunicación HTTP.

2. El servidor web recibe la solicitud y envía una respuesta al navegador que sólo muestra el contenido en HTML, los servidores web después de procesar una solicitud devuelven sólo HTML que puede incluir Javascript.

3. El usuario llena su información y hace clic en el botón “Registrarse”. El navegador por detrás recolectará esta información, y se envía los datos al servidor.

4. La solicitud llega al servidor Web, y se ejecutará el código de servidor PHP, que se conectará con la base para verificar si existe el usuario y si la contraseña coincide con el enviado por el usuario.

5. Si el usuario y la contraseña son válidos, el código del servidor, redireccionará la solicitud a otra página, la cual se conecta nuevamente a la base de datos para traer todos los datos actualizados del usuario, después de procesar la página, el servidor envía la respuesta, sólo HTML, al usuario.

6. El usuario ve en una página las últimas actualizaciones. (3)

1.6.2- Aplicación de escritorio

Una aplicación de escritorio no es más que un programa de computadora que se utiliza como herramienta para una operación o tarea específica, la cual no requiere de un sitio web para que el usuario pueda acceder a ella. La razón más frecuente para la creación de una aplicación de escritorio es la necesidad de resolver un problema o simplificar una operación compleja dentro del mismo grupo de trabajo. A continuación se muestra un ejemplo de arquitectura para este tipo de aplicaciones:

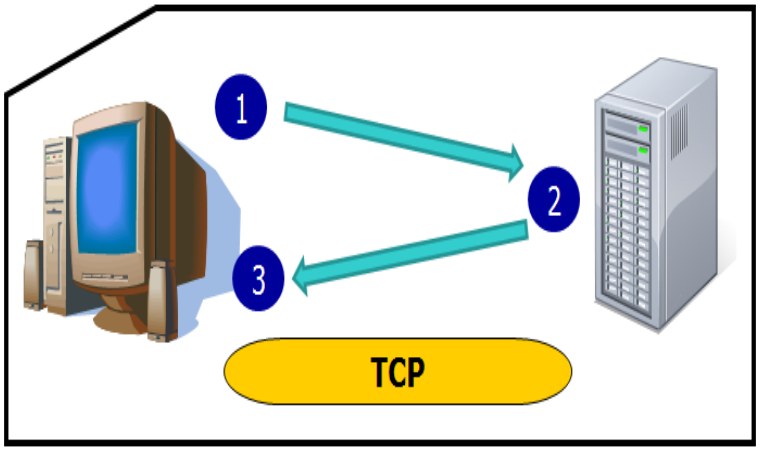

Fig.2. Arquitectura de una aplicación de escritorio. (4)

En una aplicación de escritorio normalmente no se inicia una sesión por cada aplicación que se utilice, sólo se inicia sesión una vez cuando se prende el sistema operativo, asumiendo que se va a abrir una aplicación para ver una lista de tareas:

1. El usuario carga la aplicación.

2. La aplicación, se conecta a la base de datos y recupera la información del usuario.

3. La aplicación muestra al usuario la información solicitada.

Después de analizar lo antes planteado es necesario conocer que las aplicaciones de escritorio se desarrollan para cubrir necesidades específicas de la empresa, como la contabilidad, gestión de personal, cantidad de recursos, etc. Estos son datos importantes a tener en cuenta a la hora de realizar la estimación. Además se necesita de una aplicación que tenga un tiempo pequeño de respuesta a la hora de digitalizar la cantidad de recursos y esfuerzo utilizado. A continuación se relacionan algunas ventajas de este tipo de aplicación:

- Habitualmente su ejecución no requiere comunicación con el exterior, sino que se realiza de forma local. Esto repercute en mayor velocidad de procesamiento, y por tanto en mayores capacidades a la hora de programar herramientas más complicadas o funcionales.
- Suelen ser más robustas y estables que las aplicaciones web.
- Rendimiento: el tiempo de respuesta es muy rápido.
- Seguridad: pueden ser muy seguras pues no están conectadas a una red, por lo que no se puede atacar desde otro sitio. (4)

Herramienta a emplear

La herramienta a desarrollar es para un proyecto pequeño y necesita solo de una computadora, después de observar la figura 1 se llegó a la conclusión que una aplicación de escritorio satisface el problema planteado en el presente trabajo, además las aplicaciones web, habitualmente ofrecen menos funcionalidades que las aplicaciones de escritorio.

Una vez concluido el análisis sobre los tipos de aplicaciones se decidió realizar una aplicación de escritorio debido a que el grupo de calidad del centro CEGEL es pequeño y cuenta con pocos recursos, además utilizando este tipo de aplicación no hay necesidad utilizar servidor.

1.7- Metodología de desarrollo

Según Somerville *"La metodología de desarrollo de software se encarga de elaborar estrategias; centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega. Su principal objetivo es elevar la calidad del software a través de un mayor control sobre el proceso".* (5)

Dicho de otra forma no es más que desarrollar un producto con mayor calidad. Las metodologías se deben adaptar de acuerdo a las características de cada proyecto y se clasifican en metodologías tradicionales y ágiles.

1.7.1- Metodologías tradicionales

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

Estas metodologías imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. (6)

Las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar. Por el grado de complejidad que presenta dicha metodología no es eficiente en aplicaciones pequeñas como la que se desea implementar.

1.7.2- Metodologías ágiles

Un modelo de desarrollo ágil, generalmente es un proceso incremental, basado en entregas con ciclos rápidos, también cooperativo porque clientes y desarrolladores trabajan constantemente con una comunicación muy fina y constante (7). Las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que harán la entrega del proyecto menos complicada y más satisfactoria. A continuación se relaciona una tabla con algunas características de las metodologías agiles más usadas en la universidad.

Aspectos	Scrum	XP	SXP
Cambios en las iteraciones	Los equipos no permiten cambios en sus iteraciones. Una vez que la reunión de planificación de iteración se ha completado y se ha contraído un	Los equipos son más susceptibles al cambio dentro de sus iteraciones, siempre y cuando el equipo no ha	No es posible introducir cambios durante las iteraciones, por lo tanto para planificar su

	compromiso con la entrega de un conjunto de elementos de la lista de tareas identificadas por el equipo SCRUM durante la planificación de las iteraciones del producto, estos se mantienen sin cambios hasta el final del Sprint.	empezado a trabajar en una característica particular.	duración hay que pensar en cuánto tiempo se puede comprometer a mantener los cambios fuera de las iteraciones.
Prioridad de desarrollo	El propietario del producto SCRUM, en este caso el cliente, prioriza la acumulación de productos, pero es el equipo quien determina la secuencia en la que se desarrollarán los elementos de la lista de tareas identificadas por el equipo SCRUM durante la planificación de las iteraciones del producto.	Los equipos trabajan en un orden de prioridad estricta determinado por el cliente.	Los equipos trabajan en un orden de prioridad estricta determinado por el cliente.
Prácticas de ingeniería	No prescribe prácticas de ingeniería	Establece prácticas de ingeniería como el desarrollo basado en pruebas, el enfoque en pruebas automatizadas, la programación en parejas, diseño simple y refactorización.	Establece prácticas de ingeniería como el desarrollo basado en pruebas, el enfoque en pruebas automatizadas, la programación en parejas, diseño simple y refactorización

Tabla 5. Características de metodologías ágiles. (8)

Se decide utilizar XP, además de las características planteadas en la tabla, principalmente porque el cliente es el mismo grupo de calidad, por lo que la interacción desarrollador-cliente será satisfactoria, esto constituye uno de los requisitos para llegar al éxito del proyecto. Además la metodología consiste en una programación rápida, resultando ideal para el presente trabajo pues mientras más rápido se desarrolle la herramienta, mejor será para el grupo de calidad.

XP (Extreme Programming)

La metodología XP constituye uno de los métodos fundamentales dentro de las metodologías agiles, las cuales llevan al extremo las buenas prácticas para la consecución de sistemas funcionales y que cumplan las características del cliente (9). XP surge como respuesta a la sobre planificación a la hora de generar software, está basada en la simplicidad, la comunicación, la retroalimentación y la refactorización de código. A continuación se relacionan dos características que influyen en el éxito de XP:

Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que al adelantarse en algo hacia el futuro, se puedan hacer pruebas de las fallas que pudieran ocurrir.

Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Para facilitar la organización del trabajo durante el desarrollo de software la metodología XP, propone varias fases:

- Fase#1: Planificación de proyecto.
- Fase#2: Fase de diseño.
- Fase#3: Fase de codificación.

Dentro de las principales ventajas que presenta la metodología XP se pueden citar:

- Es diferente de otras metodologías porque pone más énfasis en la adaptabilidad que en la previsibilidad.
- Se aplica de manera dinámica durante el ciclo de vida del software.
- Es capaz de adaptarse a los cambios de requisitos.
- Los individuos e interacciones son más importantes que los procesos y herramientas.
- Software que funcione es más importante que documentación exhaustiva.
- La colaboración con el cliente es más importante que la negociación de contratos.

- Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- La respuesta ante el cambio es más importante que el seguimiento de un plan. (7)

1.8- Lenguaje de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por las computadoras. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. (10) Entre los lenguajes más utilizados a nivel mundial se tiene a Delphi, Visual Basic, Pascal, Java, C++.

El centro CEGEL tiene definido una arquitectura para las aplicaciones de escritorio, la misma se rige por dos lenguajes diferentes para la implementación. Los lenguajes utilizados por el centro son Java y C#. Sin embargo la mayoría de los proyectos actualmente emplean el lenguaje Java, pues el lenguaje C# es privativo, o sea se necesita licencia y esta resulta costosa para la universidad, mientras que una de las ventajas que ofrece Java es su gratuidad. El presente trabajo decidió utilizar Java para el desarrollo de la herramienta, siguiendo los pasos de la mayoría de los proyectos actuales del centro. A continuación se muestra una tabla donde se relacionan dos características de importancia para el centro CEGEL de los lenguajes mencionados:

	Java	**C#**
Multiplataforma ideal	Al ser un lenguaje multiplataforma, la filosofía de Java "escribe una vez, ejecuta en cualquier parte" es loable.	Las diferencias entre las diversas versiones de Windows pueden causar problemas similares.
Costo	El lenguaje y la plataforma son de código abierto y la mayoría de las	La mayoría de las aplicaciones importantes, necesitan comprar licencias

	herramientas de desarrollo que usa Java resultan gratis.	de Visual Studio para desarrollar en .Net

Tabla 6. Características de Java y C#.

A continuación se realiza una breve descripción del lenguaje Java.

1.8.1- Java

Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos, sobre arquitecturas distintas y con sistemas operativos diversos. Además, Java fue diseñado para crear software altamente fiable, para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. (11)

Sus características de memoria liberan a los programadores de una gran cantidad de errores, se ha prescindido por completo de los punteros y la recolección de basura, elimina la necesidad de liberación explícita de memoria. El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado y las clases sólo se enlazan a medida que son necesitadas.

1.9- Entorno Integrado de Desarrollo (IDE)

Siguiendo los pasos de la arquitectura del centro CEGEL para aplicaciones de escritorio, donde se define Java como lenguaje de programación, la arquitectura plantea el uso de dos entornos de desarrollo como son Net Beans y Eclipse. Actualmente en el centro se fomenta la utilización de Net Beans para proyectos cortos. A continuación se mencionan algunas características de los IDE mencionados.

Eclipse es un IDE donde se instalan gran cantidad de plugins como módulos independientes que brindan un enfoque mucho más robusto para desarrollos de gran envergadura, siendo más complejo el trabajo debido a que la herramienta a desarrollar es sencilla.

Mientras que Net Beans es un muy buen entorno de desarrollo para aplicaciones pequeñas, brinda un entorno más agradable e intuitivo, adicionalmente a diferencia de Eclipse donde debemos instalar plugins para varias cosas, Net Beans ya viene con plugins y módulos integrados, evitándonos tener que configurar el ambiente, ofreciendo

todo el entorno listo para trabajar. A continuación se muestra una tabla comparativa entre estos IDE. A continuación se realiza una breve descripción de Net Beans, pues es el IDE con el cual se desarrollará la herramienta.

NetBeans IDE

NetBeans IDE es una aplicación de código abierto, diseñada para el desarrollo de aplicaciones haciendo uso de la tecnología Java. Además NetBeans IDE dispone de soporte para crear interfaces gráficas de forma visual. Es un producto libre y gratuito sin restricciones de uso. (12)

NetBeans IDE brinda la solución más completa para programar en Java. La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. A continuación se relacionan algunas características:

- Administración de las interfaces de usuario como menús y barras de herramientas.
- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- Poder crear proyectos para poder visualizar los archivos de manera gráfica.
- Puede funcionar con varios lenguajes de programación.

1.10- Sistema Gestor de Base de Datos

Los Sistemas de Gestión de Base de Datos (SGBD) se emplean como interfaz entre las aplicaciones, los usuarios y las bases de datos. Su objetivo principal se basa en gestionar convenientemente la información a almacenar, de manera sencilla y fiable para el usuario. Es por eso que actualmente se ha convertido en el principal instrumento de almacenamiento de información utilizado en la gestión de los sistemas informáticos (13). A continuación se realiza un análisis de los gestores de base de datos libres que pueden ser de utilidad para la herramienta del presente trabajo.

PostgreSQL

En comparación con MYSQL y SQLite es más lento en inserciones y actualizaciones, ya que cuenta con cabeceras de intersección, además consume más recursos. La sintaxis de algunos de sus comandos o sentencias no es nada intuitiva. Además, para trabajar con PostgreSQL hay que administrarlo mediante el PgAdmin, esto resulta una deficiencia ya que hay que instalar dicho programa para que funcione el gestor.

MySQL

MySQL a pesar de tener más velocidad en cuanto a las actualizaciones, un gran porcentaje de sus utilidades no están documentadas. Además los privilegios para una tabla no se eliminan automáticamente cuando se borra dicha tabla y presenta limitaciones para manejar la integridad referencial con tablas no transaccionales.

SQLite

SQLite es bien conocido como una base de datos que puede mantenerse en línea por periodos prolongados. Requiere de poco mantenimiento en muchos casos. Además SQLite tiene una pequeña memoria y solo necesita una biblioteca para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas. Otro aspecto importante es que este gestor, se ejecuta en muchas plataformas y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración.

Para seleccionar el gestor de base de datos se tuvieron en cuenta las características del presente trabajo y se definió que SQLite resulta más provechoso, principalmente porque ofrece la ventaja de ser un gestor portable, o sea se puede ejecutar desde cualquier computadora, tenga o no tenga servicios de administración para base de datos. A continuación se muestra una breve descripción de SQLite.

1.10.1- SQLite

A diferencia del sistema de gestión de bases de datos cliente-servidor, SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las

llamadas a funciones son más eficientes que la comunicación entre procesos. Una de las características fundamentales de SQLite es que en lugar de asignar un tipo a una columna como en la mayor parte de los sistemas de bases de datos SQL, los tipos se asignan a los valores individuales. Por ejemplo, se puede insertar un string en una columna de tipo entero, a pesar de que SQLite tratará en primera instancia de convertir la cadena en un entero. Algunos usuarios consideran esto como una innovación que hace que la base de datos sea mucho más útil.

1.11- Patrones

Un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos. En teoría, indica la manera de utilizarlo en circunstancias diversas. Para desarrollar una aplicación informática es necesario conocer bien que patrones se deben utilizar a la hora del diseño y la arquitectura. (14)

1.11.1- Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Estos utilizan un conjunto de buenas prácticas del diseño orientado a objetos para crear sistemas reutilizables y fáciles de mantener. Además permiten que el diseño orientado a objetos sea más flexible y extremadamente reutilizable. (14)

Dentro de los patrones de diseño se encuentran los patrones GRASP (Patrones Generales de Software para Asignar Responsabilidades) y los patrones GOF (Pandilla de los Cuatro).

Patrones GRASP

La función principal de este tipo de patrones es que describen un conjunto de principios fundamentales para la asignación de responsabilidades. Los patrones que forman parte de este grupo son:

Experto: Asigna una responsabilidad a aquella clase que cuenta con la información necesaria para cumplir la responsabilidad.

Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos.

Alta cohesión: Asigna una responsabilidad de forma tal que la cohesión siga siendo alta.

Bajo acoplamiento: Asigna una responsabilidad para conservar bajo acoplamiento.

Controlador: Asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase.

Patrones GOF

Los patrones de diseño según The Gang of Four (GOF) "describen soluciones simples y elegantes a problemas específicos en el diseño de software orientado a objetos".

Este tipo de patrones según su propósito se clasifican en las siguientes categorías:

Patrones estructurales: Separan la interfaz de la implementación, además, plantean las relaciones entre clases, las combinan y forman estructuras mayores. Como ejemplo se tienen los patrones Bridge, Decorator, etc.

Patrones de comportamiento: Describen la comunicación entre objetos y clases. Dentro de los más conocidos se destacan Interpreter, State, Strategy, etc.

1.12- Conclusiones parciales

Al finalizar el presente capítulo se puede arribar a las siguientes conclusiones:

- Se presentó un marco conceptual e informativo que permite lograr un mayor entendimiento del contexto.
- La selección de la metodología XP permitió la planificación y ejecución de las actividades a cumplir durante el desarrollo de la herramienta a implementar.
- Las herramientas y tecnologías actuales de desarrollo de software seleccionadas sentaron las bases para el desarrollo de la herramienta las mismas permitirán reducir el tiempo y el esfuerzo en el grupo de calidad del centro CEGEL.

CAPÍTULO II: Descripción de la propuesta de solución

2.1- Introducción

En este capítulo se abordan elementos importantes que forman parte de la solución. Se modela el negocio y se realiza la captura de requisitos funcionales y no funcionales de la herramienta. Se definen sus características siguiendo los pasos de la metodología seleccionada. Se realiza el diagrama de clases y el modelo de datos referente a las clases de la herramienta, se ofrece una explicación de los patrones empleados y se muestran ejemplos de estándares para mejorar el desarrollo de la herramienta.

2.2- Descripción del negocio

En el grupo de calidad, cuando se procede a realizar las pruebas a un software, el asesor del grupo realiza una reunión previa con los demás trabajadores, donde se tienen en cuenta elementos importantes para que el software se pruebe de la mejor manera posible (28). Se planifican los recursos con que cuenta el grupo y se distribuye el personal para la realización de las pruebas. Además dependiendo de esta planificación se realizan las estimaciones del tiempo aproximado en que se debe probar el software, conociendo la estimación del tiempo, entonces se estima cuantos especialistas deben probar para concluir el trabajo en el tiempo previsto.

2.3- Historias de Usuario

Las Historias de usuario (HU) es la técnica empleada en la especificación de los requisitos del software. Son tarjetas donde el cliente realiza una descripción breve de las características que debe tener la herramienta, sean requisitos funcionales o no funcionales. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. (9)

En la metodología XP, la comunicación desarrollador-cliente es fundamental, mediante la entrevista con el usuario, se definieron los requisitos para desarrollar la herramienta. A continuación se mencionan los requisitos identificados.

Requisitos Funcionales

RF1- Adicionar Aplicación.

RF2- Modificar Aplicación.

RF3- Eliminar Aplicación.

RF4- Adicionar Documento.

RF5- Modificar Documento.

RF6- Eliminar Documento.

RF7- Adicionar Factor de valor agregado.

RF8- Modificar factor de valor agregado.

RF9- Eliminar factor de valor agregado.

RF10- Estimación del esfuerzo para las aplicaciones.

RF11- Estimación del tiempo para las aplicaciones.

RF12- Estimación del esfuerzo para la documentación.

RF13- Estimación del tiempo para la documentación.

RF14- Calcular factor de valor agregado para las pruebas.

RF15- Gestionar pruebas de software.

RF16- Generar reporte de aplicación.

RF17- Generar reporte de documentación.

RF18- Generar reporte de estimación de tiempo y esfuerzo.

RF19- Modificar usuario.

Requisitos No Funcionales

Los Requisitos No Funcionales se agruparon según su clasificación.

Usabilidad

RNF1- La herramienta desarrollada es de tipo escritorio.

RNF2- Para que la herramienta corra, hay que instalar el JDK 8.

RNF3- La PC debe tener como mínimo 512 megabytes de memoria RAM.

Confiabilidad

RNF4- Si se interrumpe la energía a la PC, la herramienta vuelve a abrir con los datos con que se estaban trabajando.

Eficiencia

RNF5- La herramienta puede alojar infinitas cantidades de artefactos añadidos.
El trabajo se desglosó según se plantea en las fases de la metodología XP.

2.3.1- Fase#1: Planificación de proyecto

En esta fase, los clientes plantean a grandes rasgos las historia de usuario que son de interés para la primera entrega del producto, las cuales representan una breve descripción del comportamiento de la aplicación. A continuación se describen algunas de ellas en un documento entregado al cliente:

Historias de Usuario	
Nombre: Adicionar Aplicación	
Número: 1	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 1
Descripción: Esta funcionalidad permite adicionar un artefacto dado su nombre, cantidad de puntos de función y clasificación.	

Tabla 7. HU: Adicionar Aplicación.

Historias de Usuario	
Nombre: Adicionar Factor de Valor Agregado	
Número: 7	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 1
Descripción: Esta funcionalidad permite adicionar el factor de valor agregado dado su	

nombre, el peso y la complejidad.

Tabla 8. HU: Adicionar Factor de Valor Agregado.

Historias de Usuario	
Nombre: Estimación del esfuerzo para las aplicaciones	
Número: 10	**Usuario:** Desarrollador
Dependiente: No	
Prioridad: Alta	**Puntos Estimados:** 1
Descripción: Esta funcionalidad permite calcular el esfuerzo para las aplicaciones dado la cantidad de artefactos de complejidad alta, media y baja y el tiempo de duración de cada complejidad.	

Tabla 9. HU: Estimación del esfuerzo para las aplicaciones.

Reuniones diarias

Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, soluciones e ideas de forma conjunta. Las reuniones tienen que ser fluidas y todo el mundo tiene que tener voz y voto.

Estimación de esfuerzo por historias de usuario

Para hacer la medición, los programadores deben realizar una estimación del esfuerzo necesario para dar cumplimiento a la prioridad que tiene cada historia de usuario, siendo el cliente el responsable de otorgarle la prioridad a cada una de las historias de usuario. A continuación se muestra cada estimación realizada para cada una de las historias de usuario identificadas:

Número	Historias de usuario	Estimación del esfuerzo (días)
1	Adicionar Aplicación	1
2	Modificar Aplicación	1
3	Eliminar Aplicación	1

4	Adicionar Documento	1
5	Modificar Documento	1
6	Eliminar Documento	1
7	Adicionar Factor de valor agregado	1
8	Modificar factor de valor agregado	1
9	Eliminar factor de valor agregado	1
10	Estimación del esfuerzo para las aplicaciones	1
11	Estimación del tiempo para las aplicaciones	1
12	Estimación del esfuerzo para la documentación	1
13	Estimación del tiempo para la documentación	1
14	Calcular factor de valor agregado para las pruebas	1
15	Gestionar pruebas de software	2
16	Generar reporte de aplicación	1
17	Generar reporte de documentación	1
18	Generar reporte de estimación de tiempo y esfuerzo	2
19	Modificar usuario	1

Tabla 10. Estimación del esfuerzo por HU.

Duración de las iteraciones

Esta fase incluye varias iteraciones sobre la herramienta antes de ser entregada. Las iteraciones son fases de la implementación donde se obtienen resultados en un tiempo estimado (9). Una vez definidas las historias de usuario y estimado el esfuerzo propuesto para la realización de cada una de ellas, el desarrollo de la herramienta se distribuyó en 3 iteraciones con el objetivo de facilitarle el tiempo de trabajo al grupo de desarrolladores, el cual está conformado por una persona.

La duración total de las iteraciones, constituye una manera de controlar todas las tareas para que se realicen en el tiempo del que dispone la iteración. Esto facilita el orden para realizar el trabajo.

Historias de Usuario	1ra Iteración	2da Iteración	3ra Iteración	Duración (días)
Adicionar Aplicación	x			9
Modificar Aplicación	x			
Eliminar Aplicación	x			
Adicionar Documento	x			
Modificar Documento	x			
Eliminar Documento	x			
Adicionar Factor de valor agregado	x			
Modificar factor de valor agregado	x			
Eliminar factor de valor agregado	x			
Estimación del esfuerzo para las aplicaciones		x		5
Estimación del tiempo para las aplicaciones		x		
Estimación del esfuerzo para la documentación		x		
Estimación del tiempo para la documentación		x		
Calcular factor de valor agregado para las pruebas		x		
Gestionar pruebas de software			x	7

Generar reporte de aplicación			x	
Generar reporte de documentación			x	
Generar reporte de estimación de tiempo y esfuerzo			x	
Modificar usuario			x	

Tabla 11. Iteraciones por historia de usuario.

2.3.2- Fase del diseño

La metodología XP sugiere que hay que conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible e implementable que a la larga costará menos tiempo y esfuerzo desarrollar. Usar glosarios de términos y una correcta especificación de los nombres de métodos y clases ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código. Las tarjetas CRC son un ejemplo del diseño simple, son útiles en la comprensión y no requieren mucho tiempo en su realización.

Tarjetas CRC

El uso de las tarjetas CRC (Clases, Responsabilidades y Colaboración) permiten al programador centrarse y apreciar el desarrollo orientado a objetos olvidándose de los malos hábitos de la programación procedural clásica. Las tarjetas C.R.C representan objetos; la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad (15).

A continuación se muestran algunos ejemplos de relaciones entre las clases, las demás tarjetas CRC se archivaron en un documento entregado al cliente:

Tarjeta CRC # 1
Datos de la clase
Nombre: Artefacto

Responsabilidades	Colaboradores
Adicionar aplicación	Controlador
Modificar aplicación	
Eliminar aplicación	

Tabla 12. Tarjeta CRC: Aplicación.

Tarjeta CRC # 2	
Datos de la clase	
Nombre: Controlador	
Responsabilidades	**Colaboradores**
Generar reporte PDF	Artefacto, Usuario, Estimación, Factor, ConexionSQLite
Modificar tablas de factor de valor agregado	
Eliminar tablas de factor de valor agregado	

Tabla 13. Tarjeta CRC: Controlador.

Tarjeta CRC # 3	
Datos de la clase	
Nombre: Usuario	
Responsabilidades	**Colaboradores**
Modificar usuario	Controlador

Tabla 14. Tarjeta CRC: Documento.

Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, orientados a objetos. A continuación se muestra el

diagrama de clases de la herramienta a desarrollar el cual está compuesto por las relaciones que guardan las clases:

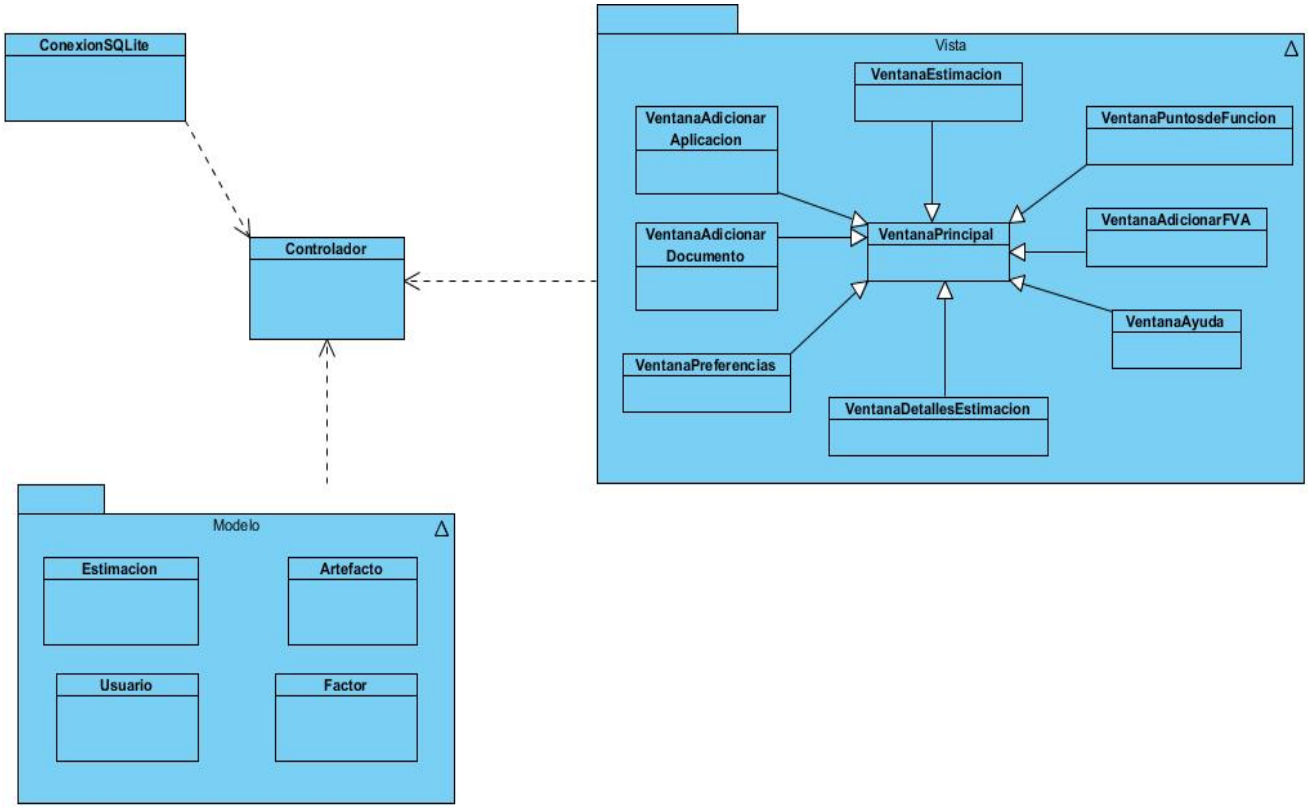

Fig.3. Diagrama de clases.

Modelo de datos

Un modelo de datos es una colección de conceptos bien definidos matemáticamente que ayudan a expresar las propiedades estáticas y dinámicas de una aplicación con un uso de datos intensivo (16). Conceptualmente, una aplicación puede ser caracterizada por:

- Propiedades estáticas: entidades, atributos y relaciones entre esas entidades.
- Propiedades dinámicas: operaciones sobre entidades, sobre propiedades o relaciones entre operaciones.
- Reglas de integridad sobre las entidades y las operaciones (por ejemplo, transacciones).

A continuación se muestra el modelo de datos correspondiente a la herramienta, el cual tiene 8 tablas en total, de ellas 4 son nomencladores y el resto es de datos:

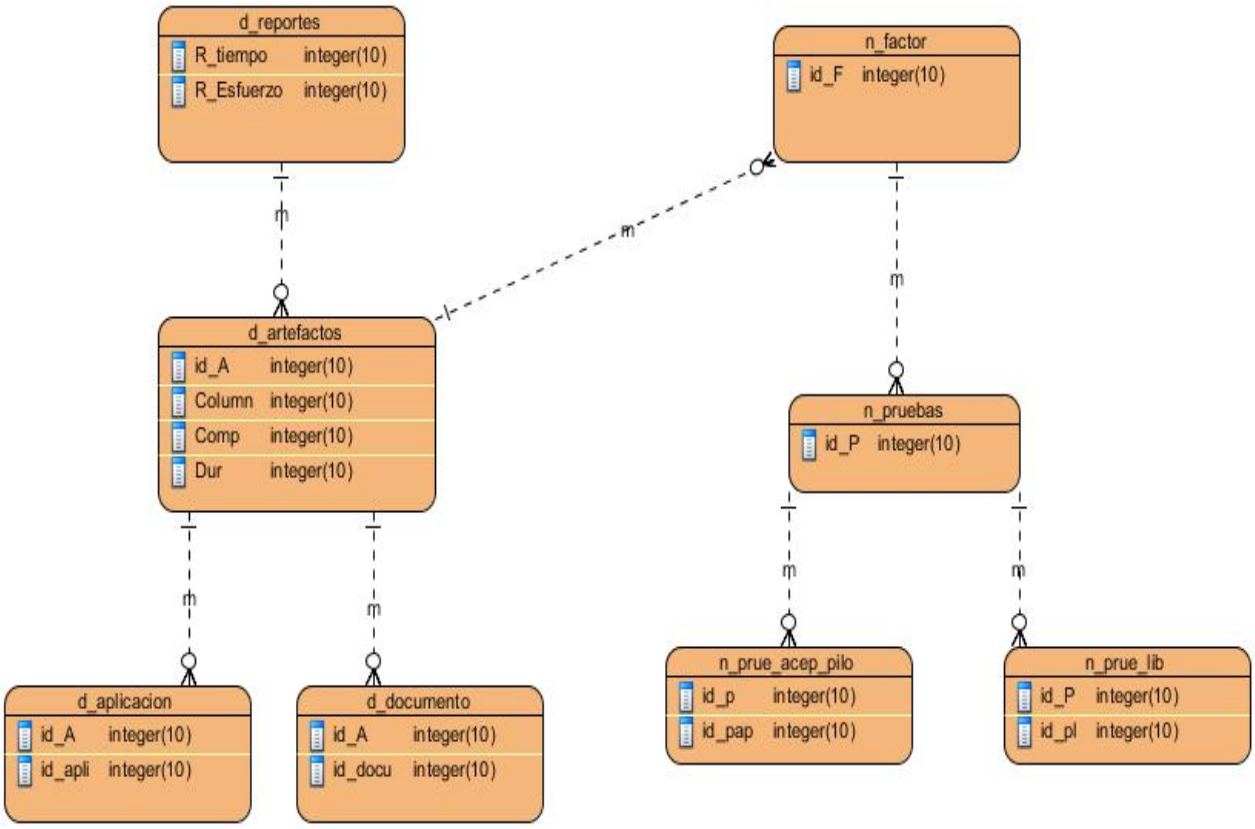

Fig. 4. Modelo de datos.

2.3.3- Fase de codificación

Una de las características de la metodología XP plantea que el cliente es una parte más del equipo de desarrollo, es muy importante su presencia en las distintas fases de XP. La codificación debe hacerse ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad. En esta fase se definen las tareas de programación o ingeniería que conlleva cada historia de usuario.

Tareas de ingeniería

Las tareas de ingeniería se utilizan con el objetivo de realizar una descripción de las tareas realizadas durante el desarrollo de la herramienta, brindado así una solución factible a las historias de usuario relacionadas a ellas (9). A continuación se muestra

una de las tareas confeccionadas, el resto se encuentran archivadas en un documento entregado al cliente:

Tarea de Ingeniería	
Número Tarea: 1	**Historia de usuario (Nro. 2):** Adicionar artefacto
Nombre Tarea: Crear interfaz Adicionar artefacto	
Tipo de Tarea : Desarrollo Desarrollo / Corrección / Mejora	**Puntos Estimados:** 1 día
Fecha Inicio: 05/3/2014	**Fecha Fin:** 06/3/2014
Programador Responsable: Osmail Blanco Pérez	
Descripción: Esta tarea tiene como objetivo crear una interfaz para gestionar los tipos de artefactos en la herramienta. Cuando el administrador de la herramienta seleccione la opción Adicionar artefacto, aparecerá la interfaz Adicionar con los campos correspondientes para crear un artefacto en la herramienta, sea de tipo aplicación o documento.	

Tabla 15. Tareas de la ingeniería.

2.4- Patrones de diseño

Los patrones de diseño son utilizados durante el proceso de desarrollo de software, estos utilizan un conjunto de buenas prácticas del diseño orientado a objetos para crear sistemas reutilizables y fáciles de mantener. Además permiten que el diseño orientado a objetos sea más flexible, elegante y extremadamente reutilizable (17). A continuación se detallan los patrones que se emplearon durante el desarrollo de la herramienta:

2.4.1- Modelo Vista Controlador

El patrón de arquitectura Modelo Vista Controlador (MVC) es un patrón que define la organización independiente del objeto del negocio, la interfaz con el usuario u otro sistema y el controlador del flujo de trabajo de la aplicación (18). De esta forma, se

divide la herramienta en tres capas donde, se tiene la encapsulación de los datos, la interfaz o vista y por último la lógica interna o controlador.

El modelo es la representación de los datos que maneja la aplicación, no tiene conocimiento específico de los controladores o de las vistas, ni siquiera contiene referencias a ellos, es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
- Define reglas de negocio.

Como ejemplo de clases desarrolladas en esta capa se tiene: Modelo.java, Documento.java y Factor.java.

La vista es el elemento que maneja la presentación visual de los datos representados por el modelo. Genera una representación visual del modelo y muestra los datos al usuario. Interactúa preferentemente con el controlador (22). Es responsable de:

- Recibir datos procesados por el controlador o del modelo y mostrarlos al usuario.
- Tener un registro de su controlador asociado.

Como ejemplo en esta capa se muestra: VentanaAdicionarAplicacion.java, VentanaAdicionarDocumento.java, VentanaEstimacionTiempoAplicaciones.java, entre otras.

El controlador es el elemento que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el modelo, centra toda la interacción entre la vista y el modelo. Maneja y entrega las solicitudes del usuario, procesando la información necesaria y modificando el modelo en caso de ser necesario (22). Es responsable de:

- Recibir los eventos de entrada.
- Implementar la lógica del negocio, es decir la funcionalidad de la aplicación.

En esta capa se desarrolló la clase: Controlador.java.

La figura que se muestra a continuación constituye un ejemplo de cómo funciona esta arquitectura:

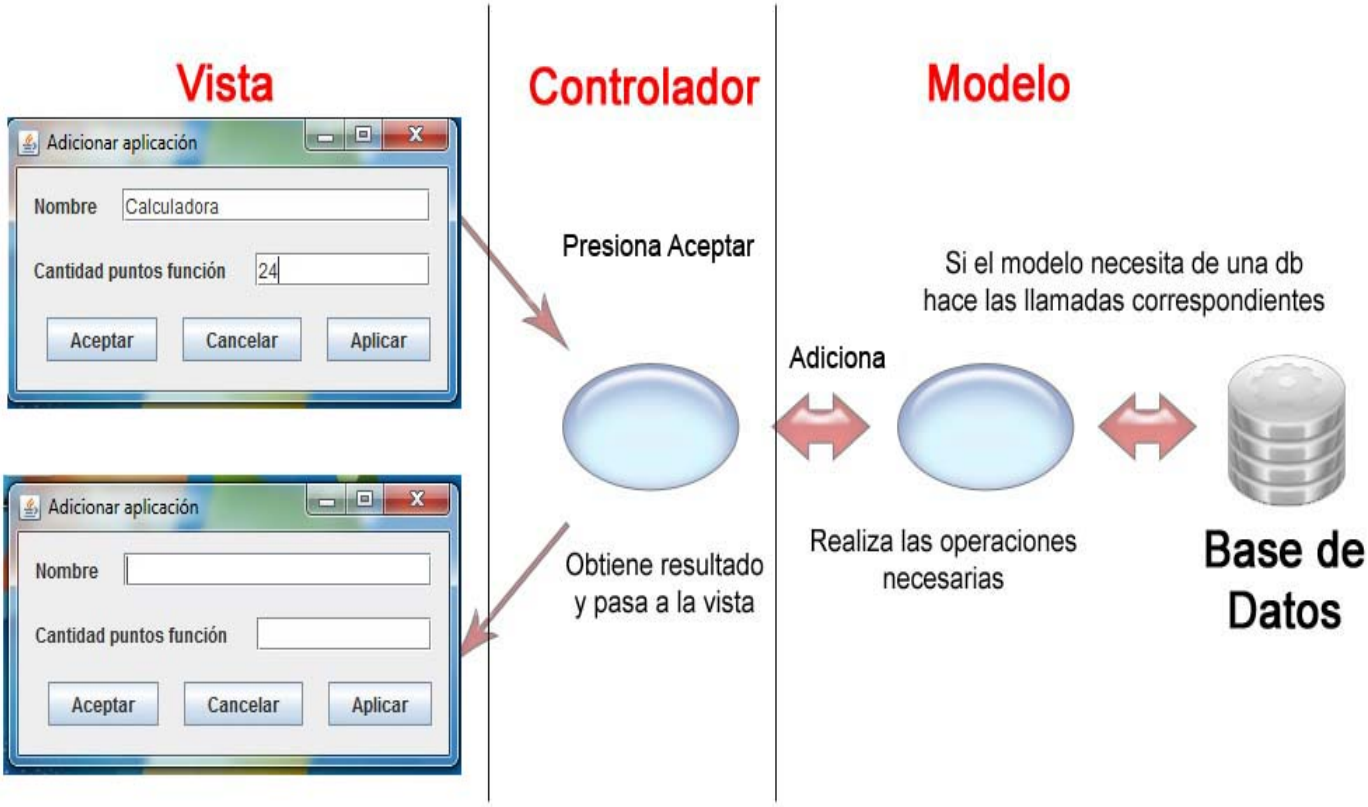

Fig.5. Modelo Vista Controlador (MVC).

2.4.2- Patrones GRASP

Controlador: Este patrón se evidencia en el presente trabajo mediante la utilización de una clase controladora donde se encuentran los métodos de mayor responsabilidad en la herramienta.

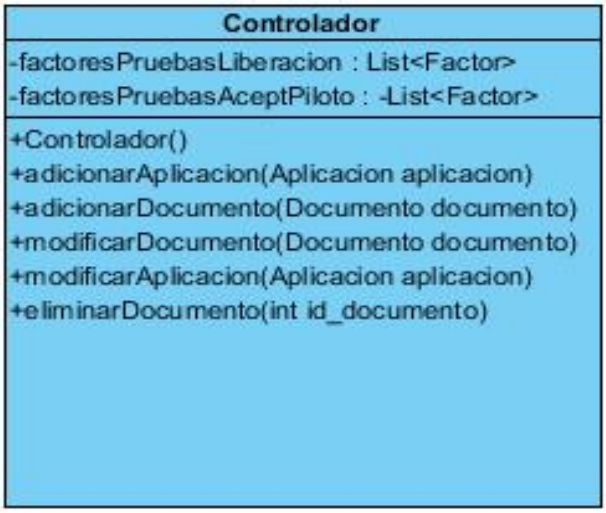

Fig.6. Ejemplo del patrón Controlador en la herramienta.

Creador: El patrón creador ayuda a identificar quién debe ser el responsable de la creación o instanciación de nuevos objetos o clases. Una de las consecuencias de emplear este patrón es la visibilidad entre la clase creada y la clase creador. Además, se tiene la información necesaria para realizar la creación del objeto que es una de las actividades más comunes en un sistema orientado a objetos y almacena o maneja varias instancias de la clase.

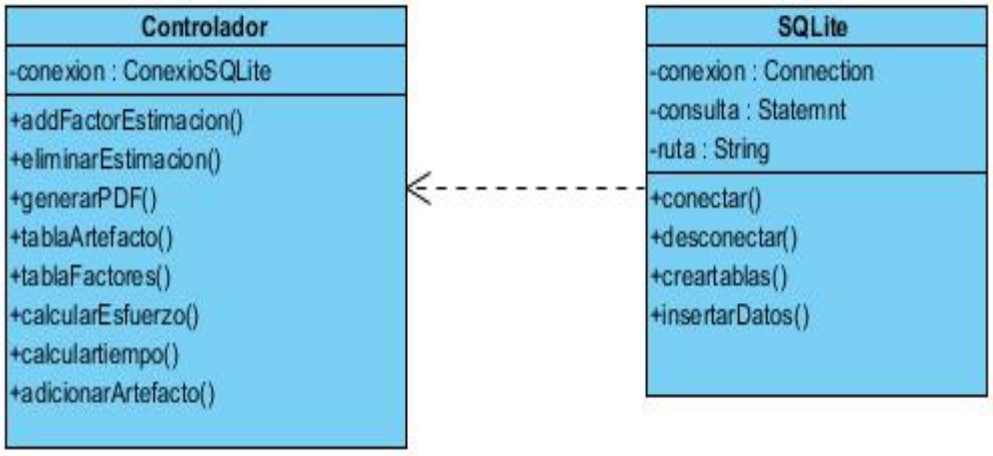

Fig.7. Ejemplo del patrón Creador en la herramienta.

Experto: La responsabilidad de realizar una labor es de la clase, que tiene o puede tener varios atributos. Una clase se considera experta si contiene toda la información necesaria para realizar la labor que tiene encomendada.

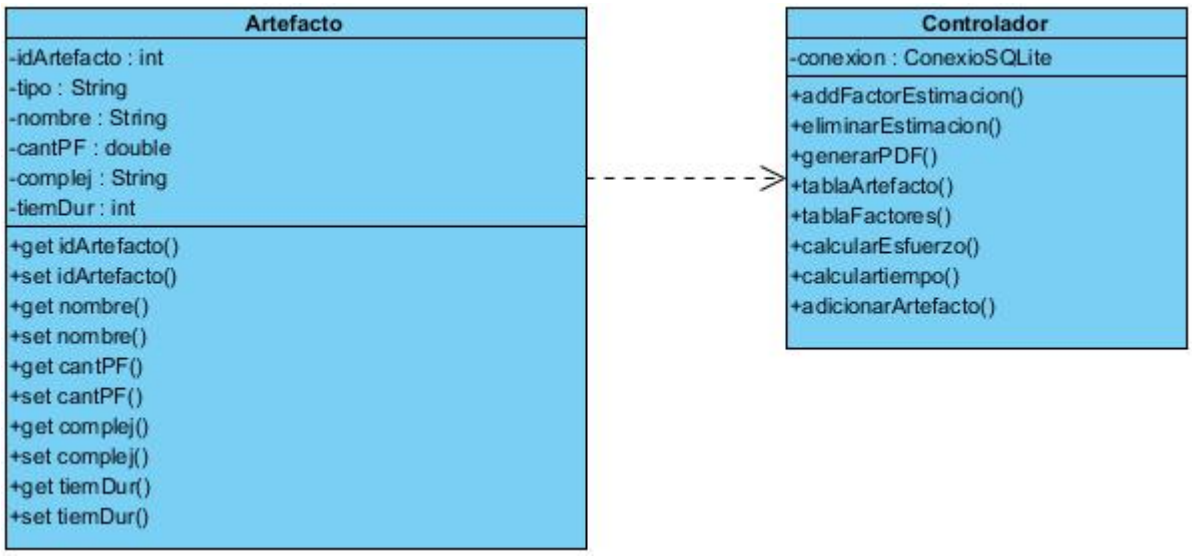

Fig.8. Ejemplo del patrón Experto en la herramienta.

Alta cohesión: La organización del trabajo es una de las características principales que se debe tener en cuenta para el desarrollo de una herramienta. Esto permite trabajar con clases que tengan una alta cohesión. En la herramienta utilizando este patrón se le asigna a cada clase las responsabilidades que le corresponden y se establecen las condiciones para que colaboren con otras.

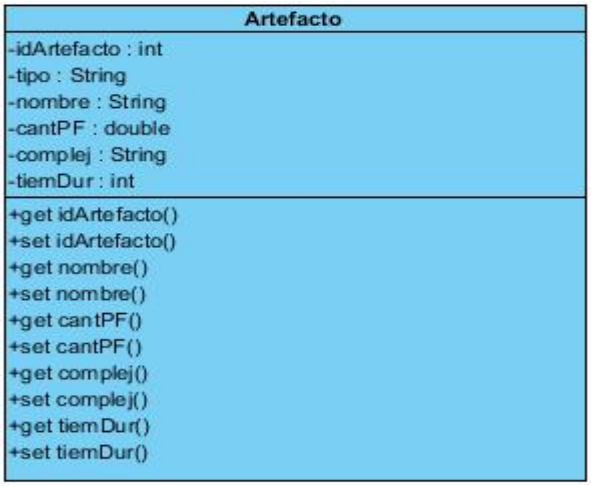

Fig.9. Ejemplo del patrón Alta Cohesión en la herramienta.

Bajo acoplamiento: Es el patrón que mantiene las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases (23):

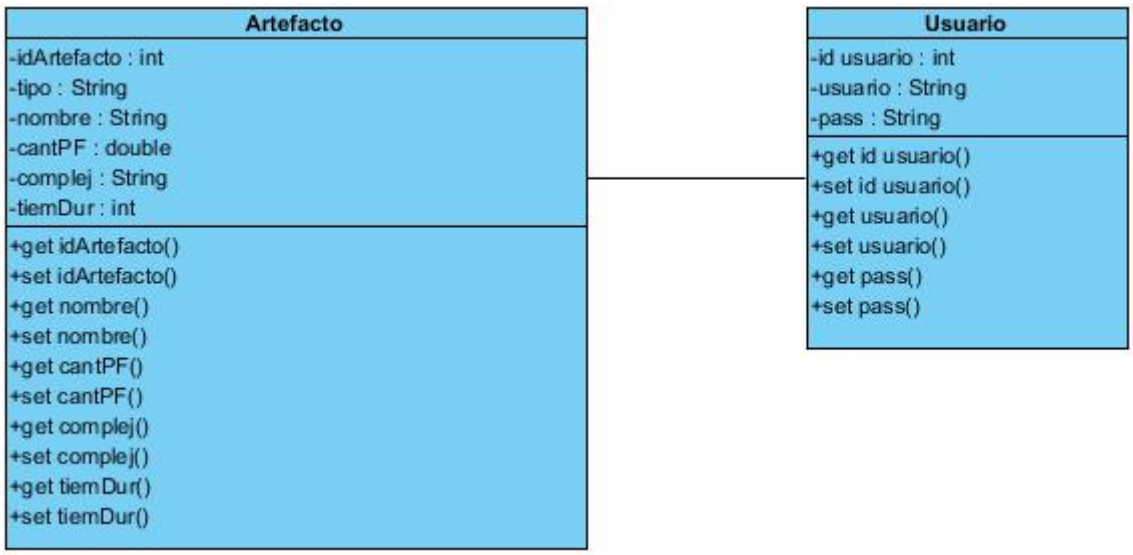

Fig.10. Ejemplo del patrón Bajo Acoplamiento en la herramienta.

2.4.3- Patrones GOF

Fabrica abstracta: El uso de este patrón en la herramienta desarrollada, proporcionó una interfaz para la creación de familias de objetos, delegando la responsabilidad de instanciarlo. Se evidencia en la clase VentanaPrincipal para la ceración de formularios, mediante el método crearPdfMenuActionPerformed ().

Decorador: Patrón estructural sirvió para añadir funcionalidades a una clase dinámicamente, se aplica en la generación de las vistas, añadiendo elementos comunes y reutilizables para varias plantillas como el menú o el pie de página (24).

2.5- Estándares de codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Al comenzar un proyecto de software, se recomienda establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Usar técnicas de codificación sólidas es de gran importancia para la calidad del software y para obtener un buen rendimiento (25). La metodología XP propone una serie de convenciones o estándares de códigos enfocados a la estructura, apariencia física de la aplicación, comprensión y mantenimiento del código. A continuación se muestran algunos estándares utilizados en el código de la herramienta:

Longitud de línea

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

```
public List<Documento> obtenerDocumentos() {
    List<Documento> documentos = new ArrayList<>();
    try {
        ResultSet resultado = conexion.obtenerDatos("SELECT * FROM documento");
        while (resultado.next()) {
            int id_documento = resultado.getInt(1);
            String nombre = resultado.getString(2);
            int cantidad_pf = resultado.getInt(3);
            String complejidad = resultado.getString(4);
            int tiempo_duracion = resultado.getInt(5);
            Documento documento = new Documento(nombre, cantidad_pf);
            documento.setId_documento(id_documento);
            documentos.add(documento);
        }
        conexion.consulta.close();
        conexion.conexion.close();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
    return documentos;
}
```

Fig.11. Ejemplo de Longitud de línea en un método de la herramienta.

Declaraciones

Es recomendable realizar una declaración por línea, ya que facilita los comentarios.

```
List<Documento> documentos = new ArrayList<>();
```

Fig.12. Ejemplo de las Declaraciones en un método de la herramienta.

Comentarios

Existen dos tipos de comentario, los de implementación y los de documentación, estos últimos existen solo en Java y se encuentran limitados por /*...*/.

```
/**
 * @return the nombre
 */
public String getNombre()
    return nombre;
}
```

Fig.13. Ejemplo de Comentarios en un método de la herramienta.

Sentencias if-else

Las sentencias if-else deben tener la siguiente forma y deben usar siempre llaves { }.

```
public final void setComplejidad()
{
    if(this.getCantidadPF() >= 1 && this.getCantidadPF() <= 10)
        this.complejidad = "Baja";
    else if(this.getCantidadPF() >= 11 && this.getCantidadPF() <= 30)
        this.complejidad = "Media";
    this.complejidad = "Alta";
}
```

Fig.14. Ejemplo de Sentencias if-else en un método de la herramienta.

Sentencias for

Las sentencias for deben ser de la siguiente manera:

```
for (int i = 0; i < aplicaciones.size(); i++) {
    if (aplicaciones.get(i).getComplejidad().equals("Alta")) {
        cantA++;
        tA += aplicaciones.get(i).getTiempoDuracion();
    }
    if (aplicaciones.get(i).getComplejidad().equals("Media"))
        cantM++;
        tM += aplicaciones.get(i).getTiempoDuracion();
    } else {
        cantB++;
        tB += aplicaciones.get(i).getTiempoDuracion();
    }
}
```

Fig.15. Ejemplo de Sentencias for en un método de la herramienta.

Variables

Todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman empiezan con su primera letra en mayúsculas. Los

nombres de variables no deben empezar con los caracteres subguión "_" o signo del dólar.

```
int cantA = 0;
int cantM = 0;
```

Fig.16. Ejemplo de declaración de Variables en un método de la herramienta.

2.6- Conclusiones parciales

Al finalizar el presente capítulo se puede arribar a las siguientes conclusiones:

- Se realizó un análisis de la herramienta ya en términos de solución, donde se proporciona una comprensión detallada de las historias de usuario, la arquitectura de la herramienta, el diagrama de clases, así como el modelo de datos.
- Se definieron los patrones de diseño empleados para lograr una mejor estructuración de la herramienta.

CAPÍTULO III: Validación de la propuesta de solución

3.1- Introducción

El presente capítulo está referido a la validación y verificación de la herramienta. Se emplean técnicas para la validación de los requisitos, con el fin de comprobar que la definición de los mismos precisa realmente lo que necesita la herramienta. Se aplican métricas de software para poder medir la calidad de los atributos del producto de forma cuantitativa, con el objetivo de evaluar la calidad durante el desarrollo de la herramienta. Además en este capítulo se verificó la calidad del resultado de la implementación, mediante los artefactos generados durante la validación de la propuesta de solución, exponiendo los resultados obtenidos por diferentes tipos de pruebas realizadas a la herramienta. Para comprobar finalmente la herramienta de estimación se realizaron pruebas de software, a través de las cuales se verificaron todos los requisitos funcionales.

3.2- Técnicas de validación de requisitos

La validación de los requisitos tiene como objetivo principal, evitar que una especificación sea mal implementada. Es fundamental comprobar antes del desarrollo del software que los requisitos se encuentran validados. Para ello se comprueba que las descripciones iniciales y la definición de los requisitos realizada tengan correspondencia, para comprobar que la herramienta funciona como el cliente desea. Para comprobar la validez de los requisitos, se llevaron a cabo las siguientes técnicas.

3.2.1- Revisión de requisitos

Las revisiones se dividieron en dos iteraciones, pues la cantidad de requisitos identificados es relativamente pequeña, en una primera iteración se detectaron tres errores relacionados con la falta de información a la hora de definir los requisitos. Para la segunda iteración se comprobó que los problemas estaban resueltos y que la definición efectuada fuera la correcta.

3.2.2- Construcción de prototipos

Para esta técnica se diseñaron prototipos de interfaz de usuario para que el cliente comprendiera como quedaría la herramienta a desarrollar. Para ello se realizaron reuniones con el cliente, donde el usuario decidió en una primera reunión, que debían cambiarse dos prototipos. Estos fueron corregidos y mostrados en una segunda reunión donde el cliente mostró su aprobación. A continuación se muestran las interfaces correspondiente al requisito ´´Adicionar aplicación´´, donde primeramente el usuario introduce el nombre del artefacto y procede a calcular los Puntos de Función, posterior a ello se muestra la interfaz para calcular los Puntos de Función y después se muestra la interfaz con el nombre y los puntos calculados. En los anexos se pueden observar otras interfaces de la herramienta.

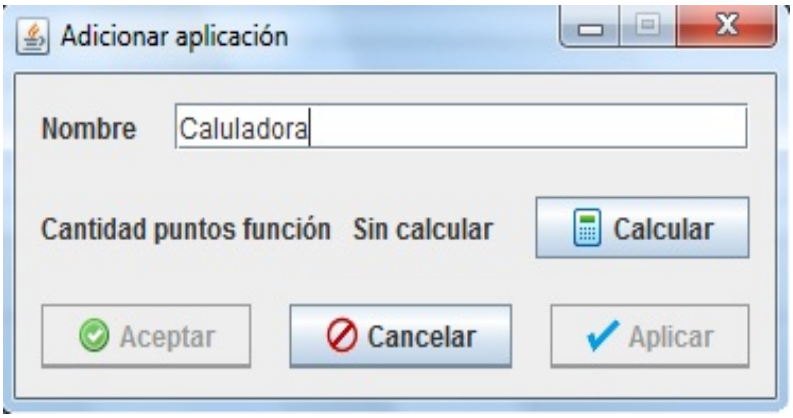

Fig.17. Interfaz para adicionar aplicación sin calcular los Puntos de Función.

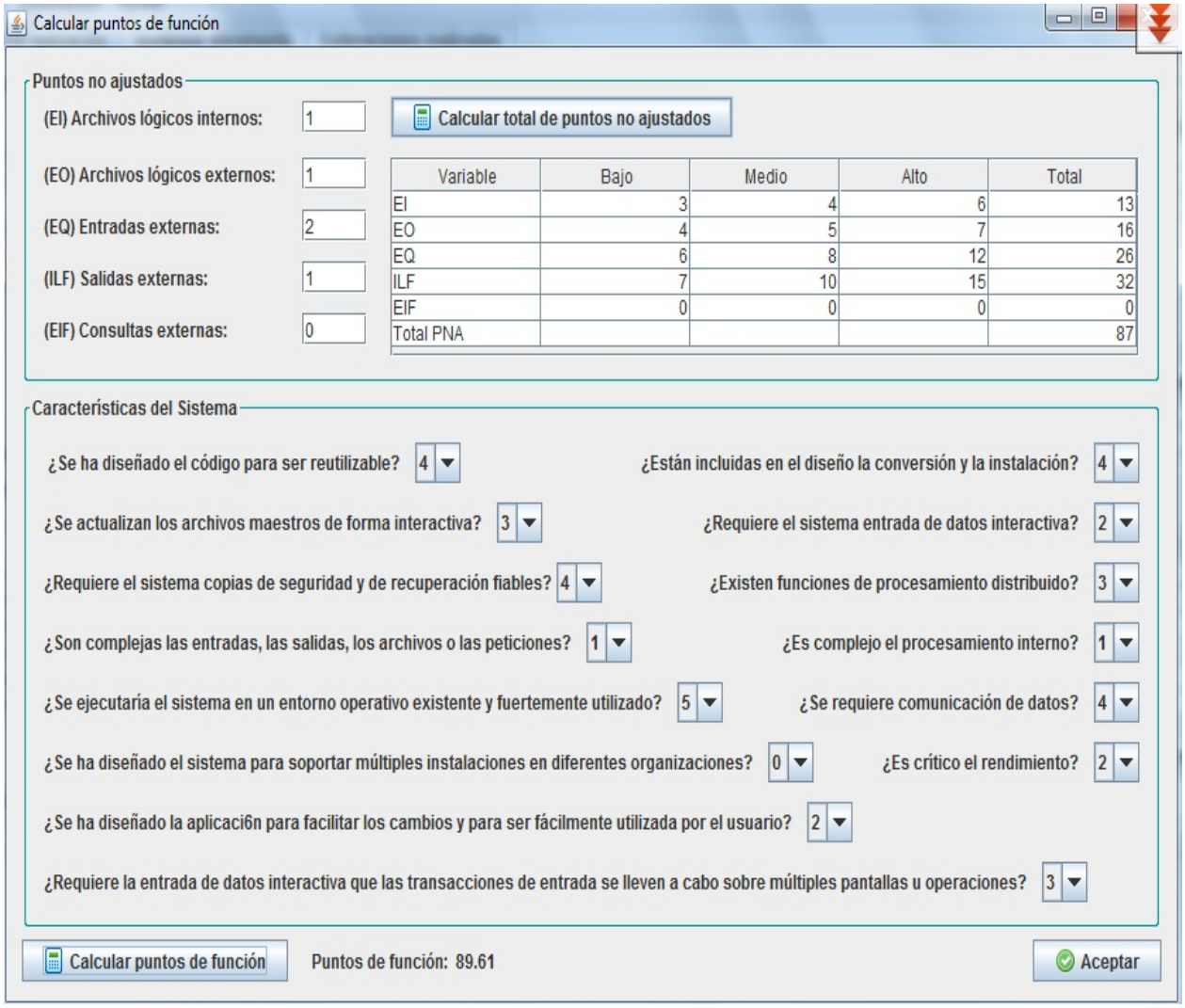

Fig.18. Interfaz para calcular Puntos de Función.

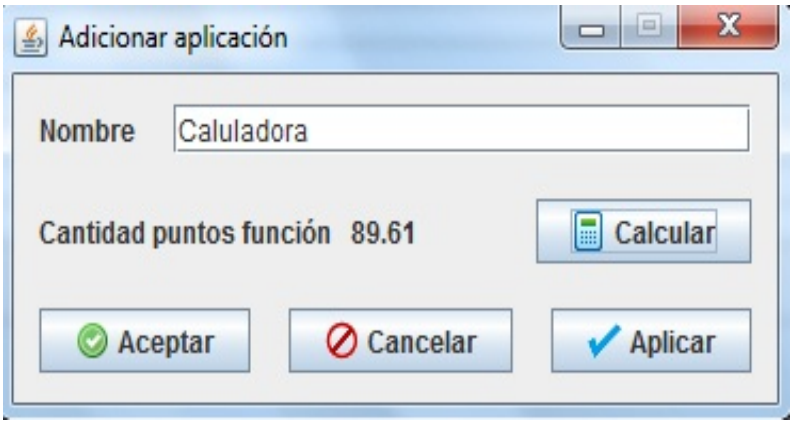

Fig.19. Interfaz para adicionar aplicación con los datos cargados.

3.3- Métricas de validación del diseño

En la construcción de la herramienta informática, se trató de cumplir con las características previstas en la fase del análisis, con el objetivo de que el cliente quedara satisfecho. Para esto se emplearon métricas, las cuales se pueden definir como la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software. Las métricas de diseño permiten medir de forma cuantitativa la calidad de los atributos internos del software. Las empleadas en la evaluación del diseño del sistema son: Tamaño Operacional de Clases y Relación entre clases. La medición persigue tres objetivos fundamentales:

- Mejorar los productos y procesos.
- Entender ¿Qué ocurre durante el desarrollo y el mantenimiento?
- Controlar ¿Qué es lo que ocurre en los proyectos?

3.3.1- Tamaño operacional de clases (TOC)

En esta medición se plantea que el tamaño total de una clase se puede determinar sumando la cantidad de atributos con el número total de funcionalidades encapsuladas en la clase. Estos valores son sumados para obtener un umbral, se le haya el promedio a este valor y se evalúan los atributos de calidad (26). Un aumento del TOC implica un aumento en la responsabilidad de las clases, así como en la complejidad de mantenimiento y una disminución del grado de reutilización de las clases puede complicar la implementación de la herramienta. A continuación se muestra un ejemplo:

	Categoria	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

Tabla 16. Rango de valores para evaluar los atributos de calidad.

A continuación se relacionan los resultados obtenidos al aplicar la métrica TOC:

Clase	Cantidad de procedimientos	Responsabilidad	Complejidad	Reutilización
Controlador	32	Media	Media	Media
Artefacto	12	Baja	Baja	Alta
Usuario	6	Baja	Baja	Alta
Factor	14	Baja	Baja	Alta
Estimación	18	Media	Media	Media

Tabla 17. Representación del tamaño de clases.

Se trabajó con un total de cinco clases las cuales arrojaron un promedio de 16,4 de cantidad de procedimientos.

A continuación se relacionan las gráficas que muestran los atributos Responsabilidad, Complejidad y Reutilización:

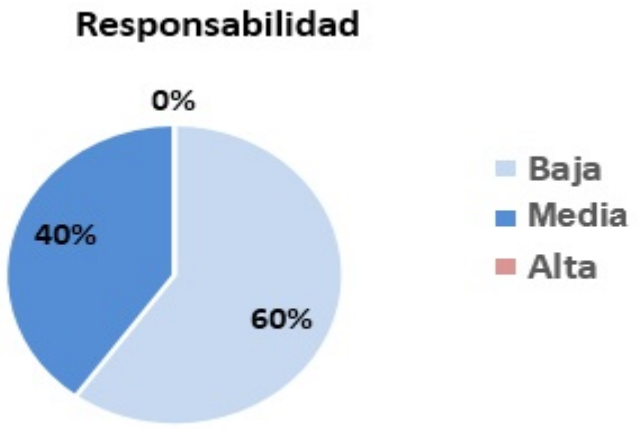

Fig.20. Porcentaje de clases agrupadas en las clasificaciones según su Responsabilidad.

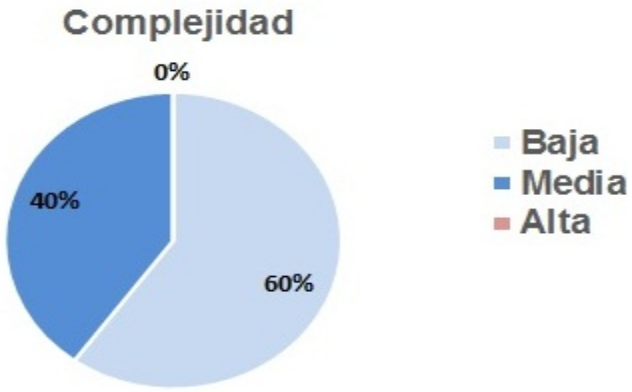

Fig.21. Porcentaje de clases agrupadas en las clasificaciones según su Complejidad.

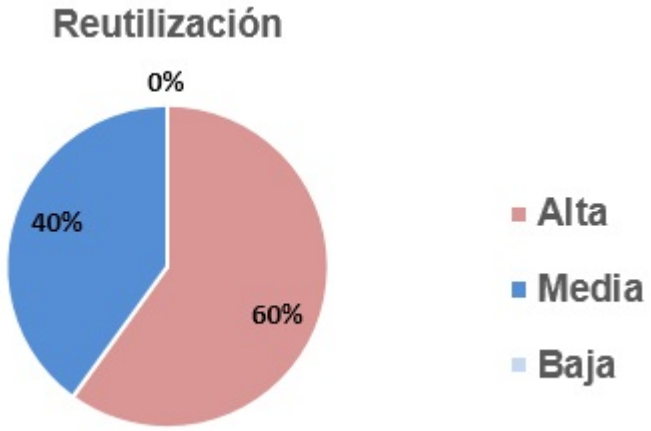

Fig.22. Porcentaje de clases agrupadas en las clasificaciones según su Reutilización.

Al analizar los resultados obtenidos para cada uno de los atributos de calidad para esta métrica, se evidencia que el 60 % de las clases poseen una responsabilidad y una complejidad baja, el 40 % de las clases poseen una responsabilidad y complejidad media y ninguna clase posee valoración alta. Mientras en la reutilización, se obtiene que el 60 % de las clases poseen una alta reutilización, el 40 % media y ninguna clase posee baja reutilización. Haciendo un análisis se puede concluir que la mayoría de las clases tienen alta reutilización y baja complejidad y responsabilidad lo cual representan resultados positivos.

3.3.2- Relación entre clases (RC)

Esta métrica está relacionada con el número de asociaciones de uso de una clase con otra. Se establece la dependencia entre dos clases cuando una clase usa métodos o variables de la otra, excepto las clases relacionadas por herencia (26). Cuanto más alto sea el acoplamiento, más difícil será reutilizarla y más riguroso ha de ser las pruebas requeridas sobre las clases involucradas. La siguiente tabla muestra los atributos que serán evaluados:

	Categoria	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

	Categoria	Criterio
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

	Categoria	Criterio
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.

	Categoria	Criterio
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

Tabla 18. Rango de valores para la evaluación de los atributos de calidad.

A continuación se muestran la evaluación de los atributos por cada clase:

Clase	Cantidad de relaciones de uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de pruebas
Controlador	0	Ninguno	Baja	Alta	Baja
Artefacto	1	Bajo	Media	Media	Media
Usuario	1	Bajo	Media	Media	Media
Factor	1	Bajo	Media	Media	Media
Estimación	1	Bajo	Media	Media	Media

Tabla 19. Resultados evaluados de la métrica RC.

La siguiente tabla relaciona los criterios de categorías de la herramienta:

Criterio	Categoría	Cantidad de clases	Promedio
0 dependencias	Muy Bueno	1	20
1 dependencias	Bueno	4	80
2 dependencias	Regular	0	0
3 dependencias	Malo	0	0
> 3 dependencias	Muy Malo	0	0

Total		5	100

Tabla 20. Criterios y categorías obtenidos en la aplicación de la métrica.

Para un total de cinco clases y un promedio de asociación de uso de 0,8 se obtuvieron los resultados siguientes:

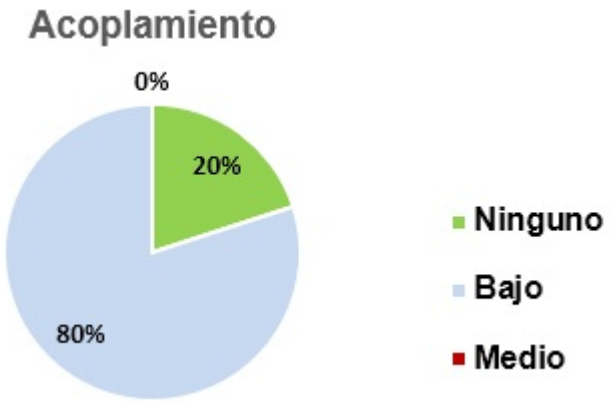

Fig.23. Porcentaje de clases agrupadas en las clasificaciones según su Acoplamiento.

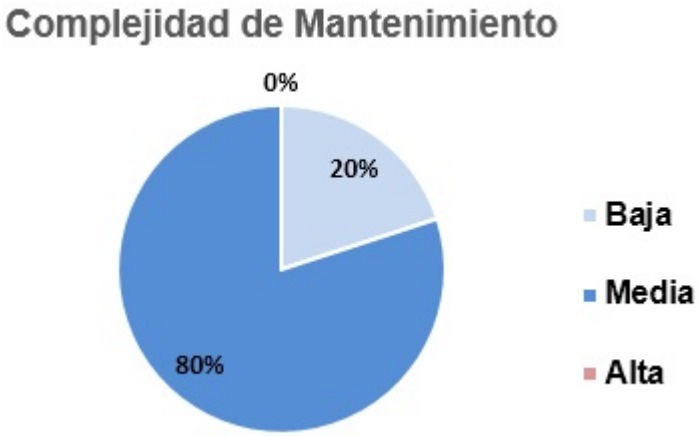

Fig.24. Porcentaje de clases agrupadas en las clasificaciones según su Complejidad.

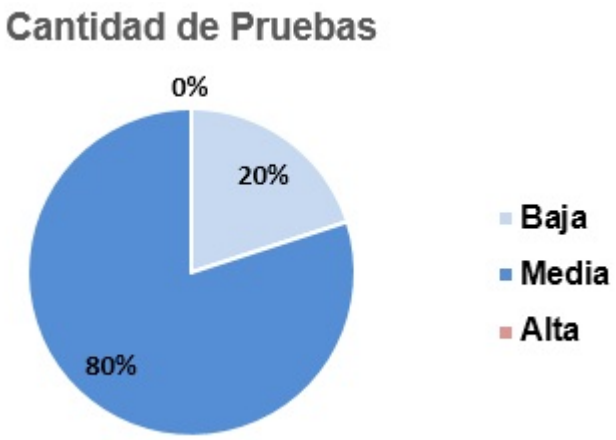

Fig.25. Porcentaje de clases agrupadas en las clasificaciones según la Cantidad de Pruebas.

Fig.26. Porcentaje de clases agrupadas en las clasificaciones según la Reutilización.

Al analizar los resultados obtenidos para cada uno de los atributos de calidad, se evidencia que el 20% de las clases no poseen acoplamiento y el 80% de ellas bajo, siendo esto muy favorable puesto que el mayor número de clases tienen bajo acoplamiento lo que implica que aumente la facilidad de reutilización de las mismas. Es por ello que en el parámetro de reutilización se puede apreciar que un 20% son de alta reutilización mientras que un 80% tienen media. Por otra parte, para la complejidad de mantenimiento y la cantidad de pruebas, de categoría baja lo representan el 20% de las clases, media el 80%.

3.4- Pruebas de software

Las pruebas de software son un proceso de ejecución de un programa con la intención de obtener errores, las pruebas constituyen un instrumento adecuado para conocer la calidad de un software. En este proceso se ejecutan pruebas dirigidas a componentes de la herramienta o a la herramienta en su totalidad, con el objetivo de medir el grado en que la herramienta cumple con los requerimientos (27). En las pruebas se usan casos de prueba, especificados de forma estructurada mediante técnicas y estrategias de prueba.

3.4.1- Estrategias de pruebas

En la metodología XP es de gran importancia el uso de las pruebas, constituye la manera de comprobar que las funcionalidades que se implementadas funcionan adecuadamente y cumplen con lo requerido por el cliente. Existen diferentes estrategias de pruebas, la cual divide las pruebas de sistema en dos grupos: pruebas de aceptación y pruebas unitarias.

Pruebas de aceptación

Las pruebas de aceptación, se realizan sobre el producto terminado, están concebidas para que sea el usuario final quien detecte los posibles errores. En esta prueba se evalúa el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto se justifique. Además las pruebas de aceptación para la metodología XP, también llamadas pruebas del cliente son especificadas por el cliente y se centran en las características y funcionalidades generales del sistema que son visibles y revisables por parte del cliente. Las pruebas de aceptación se derivan de las historias de usuarios que se han implementado como parte de la liberación del software (28).

A continuación se muestra un ejemplo de las pruebas de aceptación realizadas a historias de usuario que se mostraron en el capítulo anterior, los demás casos de prueba se encuentran archivados en un documento donde se guardan datos de las pruebas realizadas:

Caso de prueba de aceptación	
Código: HU1_F1	**Historia de usuario:** 1
Nombre: Adicionar artefacto	
Descripción: Prueba para la funcionalidad de adicionar un artefacto, ya sea de tipo aplicación o documento	
Condiciones de ejecución	
Entradas/Pasos de ejecución: El usuario elige el tipo de artefacto que desea adicionar a la herramienta, una vez seleccionado, procede a llenar los datos que se necesitan, como nombre y cantidad de puntos de función, si desea da clic en el botón aceptar y queda insertado en la herramienta el nuevo artefacto.	
Resultado esperado: La herramienta muestre en formato PDF una tabla relacionada con datos estimados sobre el artefacto adicionado**.**	
Evaluación de la prueba: Satisfactoria.	

Tabla 21. Prueba de aceptación para la funcionalidad adicionar artefacto.

La prueba de aceptación se realizó junto al cliente, donde se validó que los requisitos definidos funcionaran correctamente y cumplieran con las expectativas del usuario. Analizando los resultados obtenidos de las pruebas de aceptación, se llegó a la conclusión que el cliente quedó satisfecho, pues no se encontraron no conformidades, además la herramienta se probó con datos reales donde se obtuvieron resultados semejantes a los esperados.

Pruebas de caja blanca

Estas pruebas se centran en la estructura de control del programa. Se derivan de casos de prueba que aseguren que durante la prueba se han ejecutado todas las sentencias del programa al menos una vez y que se ejercitan todas las condiciones lógicas. Esta técnica comprueba los caminos lógicos del software, se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado sobre el código. Mediante este tipo de prueba el usuario puede:

- Garantizar que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo.

- Ejercitar todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecutar todos los bucles en sus límites.
- Ejercitar las estructuras de datos para asegurar su validez (29)

Para realizar esta prueba se decide emplear el método del camino básico, este método comprueba todos los caminos lógicos del software a través de casos de prueba, de forma que se pueda asegurar que el estado real coincide con el esperado.

Método del camino básico

El método del camino básico permite obtener una medida de la complejidad de un diseño procedimental y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

Para realizar este método se mencionan los siguientes pasos:

- Obtener el grafo de flujo, a partir del código del módulo.
- Obtener la complejidad ciclomática del grafo de flujo.
- Definir el conjunto básico de caminos independientes.
- Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores.
- Ejecutar cada caso de prueba y comprobar que los resultados son los esperados.

A continuación se muestra el código del método obtenerEstimacionesApp de la clase Controladora, el cual debe retornar una lista de las estimaciones realizadas con los factores escogidos:

```java
public List<Estimacion> obtenerEstimacionesApp()
{
    List<Estimacion> estimaciones = new ArrayList<>();
    try {
        conexion.conectar();
        ResultSet resultado = conexion.obtenerDatos("SELECT * FROM estimacion WHERE tipo_artefacto = 'Aplicacion'");
        while (resultado.next()) {
            int id = resultado.getInt(1);
            String nombre = resultado.getString(2);
            String tipo_prueba = resultado.getString(3);
            String tipo_artefacto = resultado.getString(4);
            int cantRecursos = resultado.getInt(5);
            double tiempo = resultado.getDouble(6);
            double esfuerzo = resultado.getDouble(7);
            Estimacion e = new Estimacion(null, null, nombre, tipo_prueba, tipo_artefacto, cantRecursos, tiempo, esfuerzo);
            e.setId_estimacion(id);
            estimaciones.add(e);
        }
        for(int i = 0; i < estimaciones.size(); i++)
        {
            estimaciones.get(i).setArtefactos(obtenerArtefactosDeEstimacion(estimaciones.get(i).getId_estimacion()));
            estimaciones.get(i).setFactores(obtenerFactoresDeEstimacion(estimaciones.get(i).getId_estimacion()));
        }
        conexion.desconectar();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
    }
    return estimaciones;
}
```

Fig. 27. Código del método obtenerEstimacionApp.

Una vez estudiado el método mostrado, se prosiguió a realizar el grafo correspondiente para determinar los caminos que debe tomar el método. A continuación se muestra el grafo obtenido:

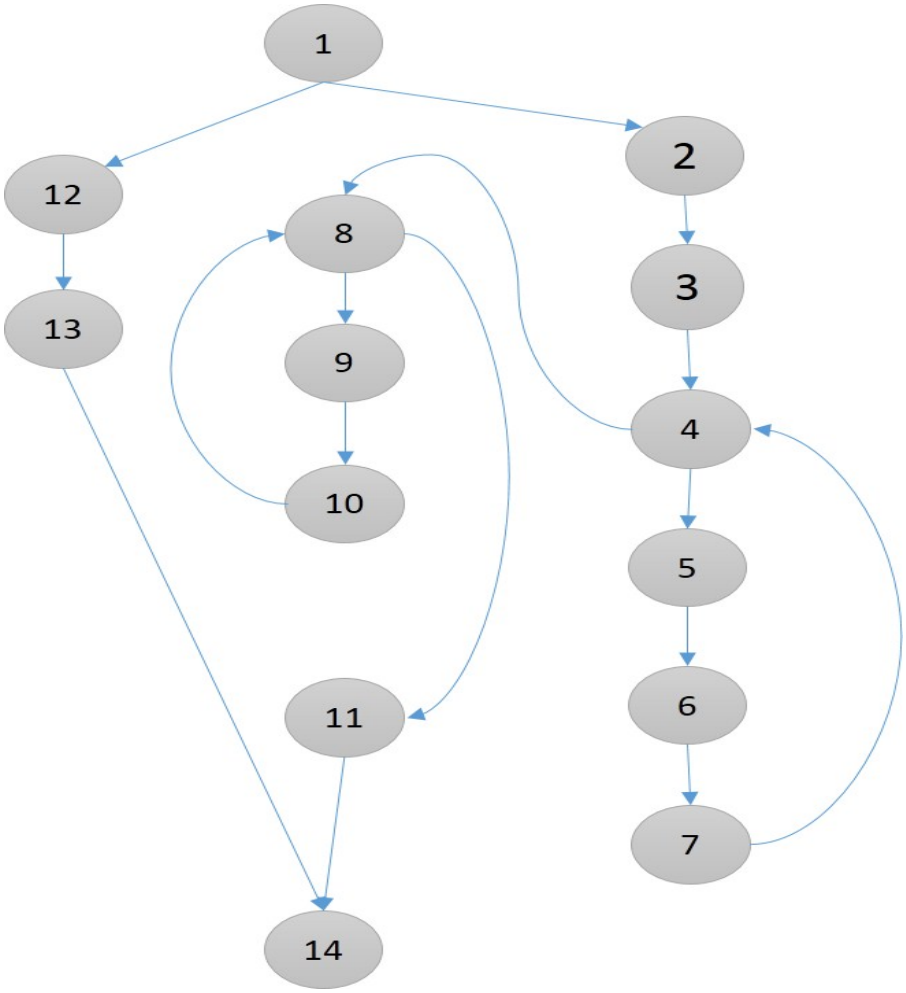

Fig.28. Grafo de flujo del método adicionar Aplicación.

Después de analizar el grafo, se calcula la complejidad ciclomática:

1: V (G)= 4

2: V (G)= (A-N+2)= 4

= (6-6+2)= 4

3: V (G)= P+1= 4

= 3+1= 4

Donde:

V (G): Complejidad ciclomática.

A: Cantidad de aristas.

N: Cantidad de nodos.

P: Cantidad de nodos predicados.

El número de caminos independientes es igual al resultado que se obtuvo de calcular V (G), en este caso V (G)= 4, por lo que quedaron definidos los siguientes caminos:

Camino 1: 1-2-3-4-5-6-7-4-8-9-10-8-11-14

Camino 2: 1-2-3-4-8-9-10-8-11-14

Camino 3: 1-2-3-4-8-11-14

Camino 4: 1-12-13-14

Al analizar el grafo de flujo e identificar los caminos a recorrer, se definieron los casos de pruebas por cada camino respectivamente, de manera que se verifique que las condiciones de los nodos predicados estén establecidas correctamente. A continuación se relacionan ejemplos de los casos de pruebas realizados, los demás se archivaron en un documento donde se guardan datos de las pruebas realizadas:

Caso de prueba para el camino 1:

Entrada	El usuario obtiene los resultados de la estimación realizada al tipo de artefacto aplicación, a partir de este se obtiene los datos relacionados con el tiempo y el esfuerzo.
Resultados esperados	Se muestra la vista con la lista de las estimaciones realizadas.
Condiciones	ResultSet resultado = conexion.obtenerDatos("SELECT * FROM estimacion WHERE tipo_artefacto = 'Aplicacion'");

Tabla 22. Caso de prueba para el camino 1.

Caso de prueba para el camino 2:

Entrada	El usuario obtiene los resultados de la estimación realizada al tipo de artefacto aplicación, a partir de este se obtiene los datos relacionados con el tiempo y el esfuerzo.
Resultados esperados	Se muestra la vista con la lista de estimaciones realizadas y además, se muestran los factores escogidos.

Condiciones	(resultado.next())

Tabla 23. Caso de prueba para el camino 2.

En una de las iteraciones de estas pruebas se encontraron dos errores, los cuales se corrigieron más tarde para garantizar que la herramienta funcione correctamente, obteniéndose resultados satisfactorios. Además se garantizó que las condiciones que pertenecen al método obtenerEstimacionesApp se establecieron correctamente.

Pruebas de caja negra

Las pruebas de caja negra son las que no toman en cuenta el código, solo necesita saber cuáles pueden ser las posibles entradas sin necesidad de entender cómo se deben obtener las salidas, donde se trata de encontrar errores en la interfaz mientras se está usando (29). Estas pruebas tienen como objetivo demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto, y que la integridad de la información externa se mantiene. En los anexos se puede visualizar algunos ejemplos de las interfaces de la herramienta a las cuales se le realizó la prueba.

Método partición de equivalencia

Este método divide el dominio de entrada de un programa en clases de datos, a partir de las cuales deriva los casos de prueba. Cada una de estas clases de equivalencia representa a un conjunto de estados válidos o inválidos para las condiciones de entrada. Los datos de entrada y los resultados de salida se agrupan en clases diferentes, en las que todos los miembros de dicha clase están relacionados. Cada una de estas clases es una partición de equivalencia en la que la herramienta se comporta de la misma forma para cada miembro de la clase (29).

A continuación se explica un caso de prueba relacionado a la técnica aplicada.

3.5- Casos de pruebas

Los casos de pruebas son un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados para cumplir un objetivo en particular. Siempre es ejecutada como una unidad, desde el inicio hasta el final. Estos casos de pruebas deben verificar si el producto satisface los requerimientos del usuario, tal y

como se describen en las especificaciones de los requerimientos. A continuación se muestra un ejemplo de los casos de pruebas realizados a la herramienta para comprobar que la funcionalidad “Adicionar artefacto” funciona correctamente, para verificar su funcionamiento se definieron dos variables, las que fueron evaluadas en tres escenarios diferentes. Los demás casos de pruebas se encuentran archivados en un documento donde se guardan datos de las pruebas realizadas.

Escenario	Descripción	Variable 1	Variable 2	Respuesta de la herramienta	Flujo central
EC1.1: Adicionar un artefacto con los datos correctos	Se llenan los campos que pide la herramienta para poder adicionar un artefacto	Nombre (Calculadora)	Puntos de Función (12)	La herramienta muestra los datos relacionados con el artefacto adicionado	Se selecciona el tipo de artefacto, se presiona el botón Adicionar y se recogen los datos a través de un formulario
EC1.2: Adicionar un artefacto con los datos vacíos	Se presiona el botón Adicionar sin llenar ningún campo	()	()	La herramienta muestra un mensaje indicando que se deben llenar los campos para poder adicionar	
EC1.3: Adicionar un artefacto con los datos incorrectos	Se ponen números en campos de letras y letras en campos de numéricos	Nombre (13)	Puntos de función (Calculadora)	La herramienta muestra un mensaje indicando que ese campo no acepta el tipo de dato escrito	

Tabla 24. Caso de prueba ´´Adicionar artefacto´´.

Resultados de las pruebas

Las pruebas se le realizaron a todas las historias de usuarios en 3 iteraciones, permitiendo encontrar varios errores. En una primera iteración se encontraron 14 no conformidades, las cuales fueron resueltas para la iteración siguiente. En una segunda iteración se encontraron 6 no conformidades, estas también se resolvieron, quedando la herramienta libre de no conformidades en la tercera iteración. A continuación se

muestra una gráfica donde se relacionan las no conformidades encontradas durante las iteraciones realizadas:

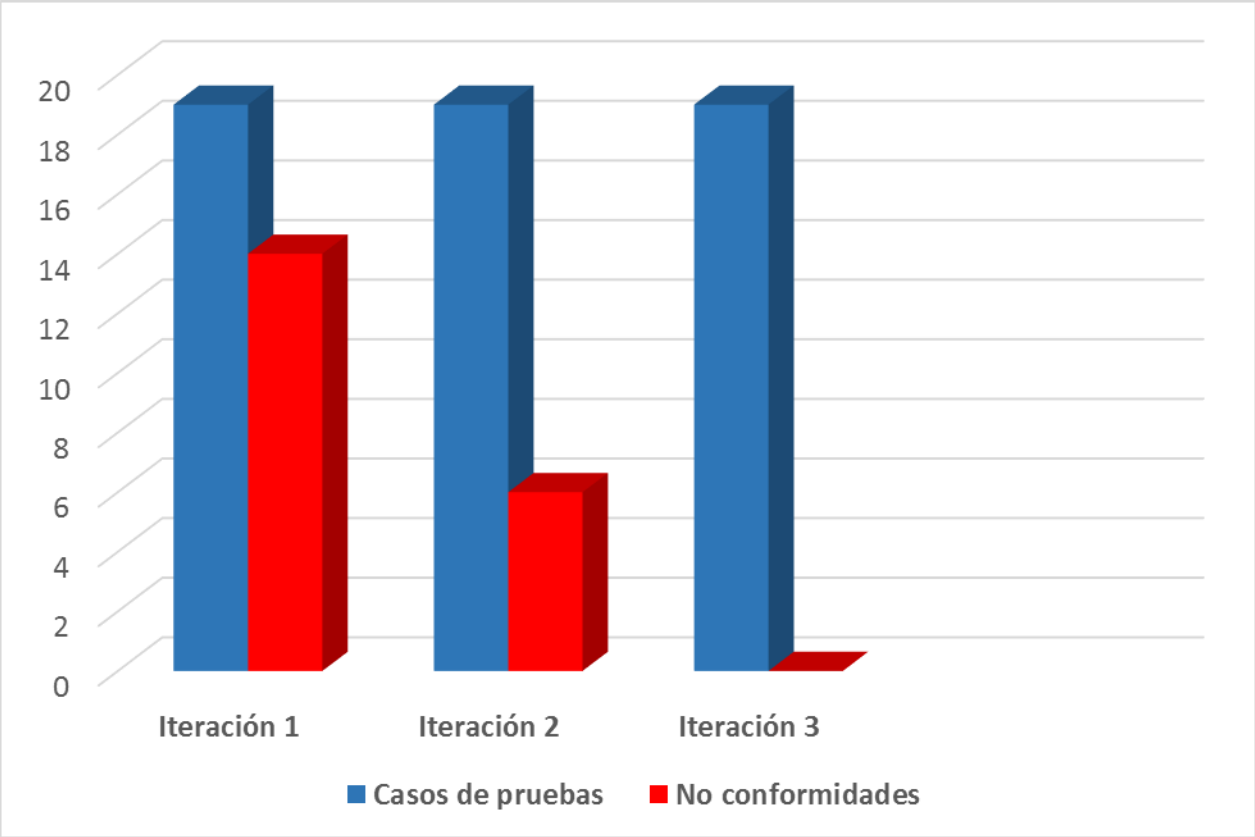

Fig.29. Gráfico de resultados de las pruebas.

3.6- Validación de las variables

Con el desarrollo de la herramienta, mediante el cálculo para estimar el tiempo y el esfuerzo en las pruebas de aceptación, liberación y piloto, se mejora el proceso de planificación de pruebas en el grupo de calidad del centro CEGEL en la UCI. A continuación se evidencia en la siguiente tabla como contribuyó la herramienta a mejorar el trabajo en el grupo de calidad:

Antes	Después
No se hacían reportes para conocer la cantidad de recursos humanos y materiales que hay que tener en cuenta	Los reportes se realizan mediante el uso de la herramienta y son emitidos en formato PDF.

para validar los productos de software que llegan al grupo de calidad.	
Para realizar los cálculos de estimación de un software que pasaba por el proceso de calidad, se necesitaban como mínimo dos personas, una se encargaba de realizar la recopilación de información sobre el software y la otra de realizar los cálculos.	Con la herramienta solo se necesita una persona que realice la estimación.
El cálculo de estimación se realizaba en hojas, por lo tanto el tiempo promedio de realizar una estimación, era aproximadamente de una hora, debido a la cantidad de fórmulas y de información.	Con la herramienta, el cálculo total de la estimación resulta sencillo y se realiza en menos de una hora.

Tabla 25. Validación de las variables.

3.7- Conclusiones parciales

Los métodos de pruebas permitieron validar el correcto desempeño de la herramienta, tanto a nivel de código, como en las funcionalidades presentes en la interfaz de usuario donde se comprobó que dichas funcionalidades fueron las esperadas por el cliente.

CONCLUSIONES

- Los métodos científicos utilizados permitieron desarrollar los conceptos y teorías que sustentan la investigación para el desarrollo de la solución.
- El estudio realizado permitió la selección de XP como metodología para guiar el proceso de desarrollo de la herramienta, se utilizó SQLite como Sistema Gestor de Base de Datos, se empleó Java como lenguaje de programación y el Netbeans en su versión 8.0 como IDE para la implementación de la herramienta de estimación de tiempo y esfuerzo para las pruebas de aceptación, liberación y piloto en el grupo de calidad del Centro CEGEL de la Universidad de las Ciencias Informáticas.
- Se realizó el diseño de la aplicación donde se elaboraron las historias de usuario y se crearon las tarjetas CRC que guiaron la implementación del software, posibilitando obtener un producto funcional capaz de satisfacer las necesidades del cliente.
- Las pruebas aplicadas durante el proceso de validación, demostraron que la herramienta cumplió satisfactoriamente con todos los requisitos planteados, demostrando así un correcto funcionamiento del software.

RECOMENDACIONES

- Proponer la utilización de la herramienta a los demás grupos de calidad de los diferentes centros productivos de la UCI.

BIBLIOGRAFÍA

1. Pressman, Roger S. Ingeniería de software (6ta edición).

2. Propuesta de un metodo de estimacion de tiempo y esfuerzo para las pruebas de aceptacion, liberacion y piloto. Santanach, Alej andro. Colombia : s.n., 2012.

3. Aplicaciones Web. Ramos, Enmanuel Valverde. Madrid : s.n., 2013.

4. Mestras, Juan Pavón. Estructura de las Aplicaciones Orientadas a Objetos. Estructura de las Aplicaciones Orientadas a Objetos. [En línea] [Citado el: 04 de 04 de 2014.] https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf.

5. Paul Mac Berthouex, Linfield C. Brown. Estadisticas para el desarrollo de la ingenieria. 2003.

6. Pruebas a traves del ciclo de vida del software. Pruebas a traves del ciclo de vida del software. [En línea] [Citado el: 24 de 03 de 2014.]

7. Antolín, Javier Uceda. Tecnologias y Servicios para la Sociedad de la Informacion . [En línea] 25 de 01 de 2005. [Citado el: 15 de 03 de 2014.]

8. Fabiola, Ticona Condori Shirley. Metodologias Tradicionales. Metodologias Tradicionales. [En línea] [Citado el: 23 de 03 de 2014.]

9. Hernán., Schenone Marcelo. Diseño de una Metodología Ágil de. Buenos Aires : s.n., 2004.

10. Canos, Jose H. Metodologias Agiles. Valencia : s.n.

11. Historia y evolución. Aranda, Vicente Trigo.

12. Aprenda Java como si estuviera en Primero. Sebastian, San. Navarra : s.n., 2000.

13. Weisser, Cristóbal. Presentación de la herramienta de programación. Presentación de la herramienta de programación. [En línea] 03 de 05 de 2000. [Citado el: 26 de 03 de 2014.] http://cristobalweisser.blogspot.com/.

14. ¿Por qué cambiarse a la Programación Orientada a Objetos? Rodríguez, Gerson Johan Samaniego.

15. Date, C. J. Introduccion a los Sitemas de Baase de Datos. Mexico : s.n., 2001.

16. Kicillof, Carlos Reynoso – Nicolás. Estilos y Patrones en la Estrategia de Arquitectura. Estilos y Patrones en la Estrategia de Arquitectura. [En línea] 24 de 03 de

2004. [Citado el: 26 de 03 de 2014.] http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF.
17. LOBOS, MARÍA ELENA DE. MÓDULO SELECCIÓN DE TÉCNICAS DE INGENIERÍA DE SOFTWARE. MÓDULO SELECCIÓN DE TÉCNICAS DE INGENIERÍA DE SOFTWARE. [En línea] 01 de 2012. [Citado el: 27 de 03 de 2014.] http://es.scribd.com/doc/216694143/Manual-Stis-2012.
18. Guía de buenas prácticas para equipos de desarrollo inexpertos en metodologías ágiles que permita trazabilidad en los requisitos asegurando Calidad del producto SW. LIZAMA, GIANNINA VALESKA COSTA. 2012.
19. Fases de la Programación Extrema. Fases de la Programación Extrema. [En línea] [Citado el: 01 de 04 de 2014.] http://programacionextrema.tripod.com/fases.htm.
20. Alfonso, Camilo SpiritualBat. Tarjetas CRC. Tarjetas CRC. [En línea] [Citado el: 01 de 04 de 2014.] http://es.scribd.com/doc/45010504/Tarjetas-CRC.
21. Tipos de datos: Clases y Objetos. Tipos de datos: Clases y Objetos. [En línea] [Citado el: 02 de 04 de 2014.] http://novella.mhhe.com/sites/dl/free/8448156315/540597/Cap_Muestra_Joyanes_8448156315.pdf.
22. Pantaleón, Marta E. Zorrilla. Modelos de datos. Modelos de datos. [En línea] [Citado el: 03 de 04 de 2014.] http://personales.unican.es/zorrillm/BasesDatos/02%20-%20Modelos%20de%20datos%20ER-UML-relacional.pdf.
23. Tedeschi, Nicolás. ¿Qué es un Patrón de Diseño? ¿Qué es un Patrón de Diseño? [En línea] [Citado el: 04 de 04 de 2014.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.
24. Patrón de arquitectura Modelo Vista Controlador (MVC). Patrón de arquitectura Modelo Vista Controlador (MVC). [En línea] [Citado el: 04 de 04 de 2014.] http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html.
25. Casallas, Gloria Cortez y Rubby. Introduccion a los patrones de diseño. Introduccion a los patrones de diseño. [En línea] [Citado el: 06 de 04 de 2014.] http://sistemas.uniandes.edu.co/~isis2701/dokuwiki/lib/exe/fetch.php?media=isis2701-patronesgrasp.pdf.

26. Patrones de diseño(GoF). Patrones de diseño(GoF). [En línea] [Citado el: 06 de 04 de 2014.] https://infow.wordpress.com/category/patrones-de-disenogof/.
27. Estándares de Codificación. Estándares de Codificación. [En línea] [Citado el: 07 de 04 de 2014.] https://docs.google.com/document/d/1rbxDFM0zsbFDNRZeM2FoXfRDbYSiSt6tCdbYPA0qdzs/edit?hl=en_US&pli=1.
28. Métricas de Software. Métricas de Software. [En línea] [Citado el: 25 de 04 de 2014.]
29. Las pruebas de software. Las pruebas de software. [En línea] [Citado el: 01 de 05 de 2014.] http://www.pruebasdesoftware.com/laspruebasdesoftware.htm.
30. casteele, john. Cómo llevar a cabo una prueba de aceptación. Cómo llevar a cabo una prueba de aceptación. [En línea] [Citado el: 01 de 05 de 2014.] http://www.ehowenespanol.com/llevar-cabo-prueba-aceptacion-del-usuario-pau-como_131118/.
31. Técnicas de prueba. Técnicas de prueba. [En línea] [Citado el: 02 de 05 de 2014.] http://indalog.ual.es/mtorres/LP/Prueba.pdf.

GLOSARIO DE TÉRMINOS

Calisoft: Centro Nacional de Calidad de Software

CEGEL: Centro de Gobierno Electrónico

CRC: Clases, Responsabilidades, Colaboración

FVA: Factor de Valor Agregado

GOF: Pandilla de los 4

GRASP: Patrones Generales de Software para Asignar Responsabilidades

HU: Historias de Usuario

IDE: Entorno Integrado de Desarrollo

MVC: Modelo Vista Controlador

PF: Puntos de Función

RC: Relación entre Clases

RUP: Proceso Unificado de Desarrollo

SGBD: Sistema Gestor de Base de Datos

TOC: Tamaño Operacional de las Clases

UCI: Universidad de las Ciencias Informáticas

XP: Programación Extrema