Universidad de las Ciencias Informáticas

Facultad 5

Herramienta de Apoyo a la Producción en el CEDIN

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

Autora: Adisleidys Mirabal Pérez

Tutor: Roberto Alejandro Espí Muñoz

La Habana, Junio 2011
"Año 53 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro que soy la única autora de este trabajo y autorizo al Centro de Informática Industrial de 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 _____.

_____	_____
Firma de la autora	**Firma del tutor**
Adisleidys Mirabal Pérez	Roberto Alejandro Espí Muñoz

DATOS DE CONTACTO

Síntesis del Tutor:

Nombre: Roberto Alejandro Espí Muñoz

Categoría Científica: Ingeniero

Especialidad: Ingeniero en Ciencias Informáticas

Correo Electrónico: raespi@uci.cu

Categoría docente: Adiestrado

Años de experiencia: 5

Años de graduado: 2

DEDICATORIA

A mi familia por confiar en mí y en especial:

A mis padres que son los ángeles de mis sueños, por poseer siempre una sonrisa agradable y sincera que me brinda la calma, el amor y la comprensión, por apoyarme siempre para alcanzar todas mis metas.

A mi hermano, que espero sepa siempre cuánto lo quiero.

A Erick Daniel, el bebé más hermoso del mundo, con la esperanza de que algún día este trabajo le sirva de inspiración y guía.

AGRADECIMIENTOS

Quiero agradecer principalmente a mi tutor, quien fuera tan buen guía como amigo, gracias por el apoyo, por la PACIENCIA, por las largas horas dedicadas a este trabajo.

A mis padres y mi hermano, que han dado tanto por mí, que me apoyaron en todas y cada una de las decisiones que tomé en mi vida, aún cuando muchas significaban lejanía entre nosotros. A ustedes que fueron pacientes, sabios consejeros en cada circunstancia. Y quienes espero recuerden siempre cuánto los amo.

A mi familia en general, gracias por confiar en mí.

A la pareja más hermosa que conozco, Yadi y Erick sepan siempre cuánto los admiro y los quiero.

A mi querida profe Hilda, gracias por enseñarme que en la vida no hay logro más placentero que el que se alcanza con el esfuerzo propio. A mis profesores Zenaida, Sailyn, Gelson, Willian y Adiel, personas que comencé a admirar desde que las conocí.

Quiero darles las gracias a todos mis amigos, por ser las personas más especiales del mundo, por estar siempre pendientes de mí, a los que conozco desde el IPVCE Juli y Madi, nada puede superar tantos tiempo de amistad que me han brindado, a los que se han unido en el transcurso de estos años y lograron ocupar un espacio de mi corazón: Danays, Yuniel, Yuneisis, Julio, Yixi y Saily. A Anay, Lisbeth y René, gracias por enseñarme que nunca es tarde para hacer nuevos amigos. Quiero agradecer de forma muy, muy especial, a mis amigos de estos 5 años: YARE no se qué sería de mí sin tus sabios consejos, sin tu paciencia a mis locuras, sino hubieses estado conmigo cuando más lo he necesitado. LETY, gracias por ser tan especial, por confiar en mí y por hacerme saber que estás ahí siempre que te necesite. Reni, siempre te dije que este título sería por los dos, gracias por la confianza, no olviden nunca que aquí estoy siempre, como estuvieron ustedes en esta larga carrera, en las que perdí tanto como lo que gané, pero siempre quedaron ustedes, gracias por eso.

Robe, eres una de las personas que más me aportó en estos 5 años, un día me dijiste que el mayor por ciento de mi formación estaría en mi tesis, hoy puedo decirte que no fue así, me formaron como ingeniera las largas noches de estudio, la sana competencia para ver quién podía enseñar al otro, competencia que nos llevaba a auto prepararnos por si el otro lo necesitaba, gracias por tus interminables críticas sobre algún trabajo mal hecho, por tu disposición a ayudarme aún cuando no compartiéramos el mismo criterio, tus consejos quedarán por siempre en mi mente, otra vez: gracias.

RESUMEN

El desarrollo de las tecnologías informáticas en el mundo, ha hecho que muchas empresas se encuentren en un proceso de restructuración, sustituyendo la mano de obra tradicional por tecnología de punta. Por esta razón, las empresas de desarrollo de software han adoptado nuevas formas de desarrollo, automatizando la mayor parte de los procesos para garantizar la calidad y las altas exigencias de los cronogramas de entrega.

En el Centro de Informática Industrial de la Universidad de las Ciencias Informáticas procesos como la integración, la aplicación de pruebas y métricas al código se realiza de forma manual, lo que provoca atrasos en el cronograma y problemas de calidad en el producto final, ya que no es posible certificar continuamente la estabilidad del producto.

Para solucionar este problema se implementó una herramienta de apoyo a la producción en el centro que permite la integración continua, la aplicación de pruebas automatizadas y verificación de métricas a código fuente.

Para el desarrollo de la solución, fueron seleccionadas las herramientas adecuadas, haciendo énfasis en el uso del software libre, así como en la selección de una metodología óptima para el proceso de desarrollo.

Palabras Clave: Integración continua, Prueba, Métrica.

ÍNDICE DE CONTENIDO

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

1

INTRODUCCIÓN

En las últimas décadas, debido al desarrollo de las Tecnologías Informáticas (TI), una gran cantidad de empresas se encuentran inmersas en un proceso de reestructuración, cambiando el uso de mano de obra tradicional por tecnología de punta, teniendo como principal elemento el uso del software. Dichos cambios han provocado que las empresas, que se dedican a la producción de software, tengan cada día más demanda; lo que trae como consecuencia que las mismas se planteen nuevas estrategias de desarrollo para cumplir con las elevadas exigencias del mundo actual.

Un proceso de desarrollo de software, tiene como principal meta la creación de un producto con calidad que satisfaga al cliente, automatizando la mayor cantidad de actividades posibles para ganar en eficiencia y evitar posibles errores cometidos por los seres humanos. Muchas de estas automatizaciones se han concentrado en la etapa de integración y en la realización de pruebas al código fuente de un producto.

En los últimos años el gobierno cubano ha realizado un gran esfuerzo por insertarse en el mercado del software, teniendo como principal exponente a la Universidad de las Ciencias Informáticas (UCI) donde se desarrollan productos y soluciones tecnológicas integrales. Dentro de la universidad se encuentra el Centro de Informática Industrial (CEDIN), que desarrolla tecnologías, productos y servicios asociados a la Informática Industria haciendo uso de las tecnologías libres, además de contribuir a la formación especializada y al desarrollo de investigaciones afines que garanticen un alto valor agregado.

Para un desarrollo eficiente y mejor aprovechamiento de las competencias de cada uno de sus miembros, el CEDIN se encuentra dividido en departamentos, entre los que se encuentra el Departamento de Integración y Despliegue, el cual no cuenta con una herramienta, que utilizando tecnologías libres, automatice el proceso de integración de código y que a su vez realice pruebas y verifique métricas de calidad del mismo, provocando que exista un tiempo prolongado entre la integración de una versión del código y la siguiente, trabajo que es realizado de forma manual. Estas condiciones de trabajo provocan atrasos en el cronograma y problemas de calidad en el producto final, ya que no es posible certificar continuamente la estabilidad del producto.

INTRODUCCIÓN

Teniendo en cuenta lo anteriormente planteado el **problema científico** a resolver queda formulado por la siguiente interrogante: ¿Cómo automatizar los procesos de integración, prueba y la verificación de métricas de calidad en el CEDIN para obtener un producto más estable?

Del problema planteado se deriva como **objeto de estudio** los procesos de integración de software así como las pruebas y métricas de calidad aplicables a código fuente, el **campo de acción** lo constituye la integración continua, pruebas y métricas de calidad al código de aplicaciones industriales.

Para dar solución a la problemática descrita se ha planteado como **objetivo general** de la investigación diseñar, implementar y desplegar una herramienta que permita la integración continua, prueba y la verificación de métricas de calidad.

De acuerdo con el problema científico **la idea a defender** es la siguiente: Con el diseño, implementación y despliegue de una herramienta que permita la integración continua, realice pruebas y verifique métricas de calidad, estos procesos se realizarán de manera eficiente y se obtendrá un código más estable en menor tiempo.

Para dar cumplimiento al objetivo general planteamos las siguientes **tareas de investigación:**

- Elaboración de la propuesta a partir del análisis de aplicaciones o soluciones existentes.
- Selección del lenguaje y metodología de desarrollo de software a utilizar.
- Selección de herramientas libres a integrar a partir de un análisis comparativo de las existentes.
- Captura de requisitos del sistema.
- Generación de artefactos ingenieriles propuestos por la metodología seleccionada.
- Implementación del software para cumplir con las funcionalidades requeridas.
- Validación de las funcionalidades del software mediante los diferentes tipos de pruebas.
- Concepción de la documentación para el uso de la herramienta.
- Despliegue de la solución para su uso en un entorno real de explotación.

INTRODUCCIÓN

Entre los métodos de trabajo científico utilizados para llevar a cabo esta investigación se destacan los siguientes:

Métodos Teóricos:

Histórico-lógico: Para realizar el estudio crítico de soluciones existentes, y utilizar estas como punto de referencia para el desarrollo de la solución.

Analítico-sintético: Permitirá descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución de la propuesta.

Métodos Empíricos:

Entrevista: Permitirá contactar personas que están involucradas en el campo de acción y que puedan aportar información valiosa para la investigación.

La investigación estará estructurada por: Introducción, tres Capítulos, Conclusiones, Recomendaciones, Referencia bibliográfica, Bibliografía, Glosario de términos y Anexos.

Capítulo 1. Fundamentación Teórica: En este capítulo se abordarán principalmente los conceptos de integración, prueba y métrica. Se realizará un estudio de las metodologías, estándares de desarrollo de software, tecnologías actuales, lenguajes de programación y herramientas, el cual permitirá la selección de las adecuadas y usarlas en el desarrollo del sistema que dará solución a la problemática actual.

Capítulo 2. Desarrollo de la propuesta: En este capítulo se describirá brevemente el flujo actual de los procesos involucrados en el problema en cuestión, haciéndose un análisis de cómo se ejecuta actualmente el proceso de integración en el CEDIN. Se detallarán los artefactos de la metodología que se propone en el capítulo anterior, quedarán definidas las tareas generadas a partir del desarrollo de las funcionalidades y los diseños de casos de pruebas para validar la solución.

Capítulo 3. Aplicación de la solución: En este capítulo se ejecutarán las pruebas diseñadas anteriormente, además quedarán descritos los aspectos fundamentales del despliegue de la aplicación, así como los resultados del uso de la herramienta por los proyectos del CEDIN.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

Introducción

Este capítulo expone los temas fundamentales que sustentan la investigación. Se mencionan principalmente los conceptos de integración, pruebas y métricas. Se abordan temas como la integración continua especificando herramientas existentes que la propician, el proceso de pruebas de software así como la importancia de la automatización de las mismas, además de la aplicación de métricas a código fuente. Se fundamentan las herramientas seleccionadas para la solución del problema.

1.1. Integración

Una de las actividades del proceso de desarrollo de software es la integración, la misma tiene como objetivo validar la lógica de los componentes o módulos internos de un proyecto de software y que además su interoperabilidad con otros externos sea correcta.

La integración de código consiste en tomar las fuentes de una entrega anterior y agregar el código fuente de las nuevas funcionalidades desarrolladas y generar el programa ejecutable y/o el instalador según sea conveniente. (Gómez 2007)

Para la integración de un sistema es necesario que se encuentren disponibles versiones anteriores de los componentes en el Sistema de Control de Versiones (SCV) del proyecto, se coordinan las actividades a realizar y todo el equipo de desarrollo participa. Conforme se van integrando componentes, se va probando que funcionen juntos adecuadamente. Se proponen las pruebas que se aplicarán para verificar la correcta interacción entre estos componentes.

El elemento de un paquete que no dependa de otros, es el primero que se integra. En el caso de las clases, una clase depende de otra si invoca alguno de sus métodos.

Al integrar los componentes se identifican los métodos que se invocan entre clases. Estos métodos deben ejecutarse para comprobar que el paso de parámetros y los resultados devueltos son los adecuados. (Gómez 2007)

La integración es uno de los procesos más difíciles dentro de un proyecto de software. Cuando muchos desarrolladores colaboran en proyectos complejos, la integración de las diferentes partes de código puede ser un proceso largo e impredecible, que al realizarse de forma convencional trae diversos problemas como son: código inestable, solo una persona que puede realizar construcciones del proyecto, el ciclo para obtener retroalimentación es muy largo, etc. Sin embargo, este proceso puede resultar más eficaz y más confiable si se realiza la integración del proyecto de manera continuada. (Microsoft 2010)

1.2. Integración continua

La **integración continua** es una práctica de desarrollo de software, propuesta inicialmente por Martin Fowler, donde los miembros de un equipo realizan integraciones frecuentes, por lo general cada persona integra al menos una vez al día, lo que conduce a múltiples integraciones diarias. Cada integración es verificada por una herramienta de construcción automatizada (incluyendo la prueba) para detectar errores de integración lo más rápido posible.

La integración continua supone un alto grado de las pruebas que se realizan automáticamente en el software (Fowler 2006). El proceso suele ser, cada cierto tiempo, descargarse las fuentes desde la herramienta de control de versiones, compilarlo, ejecutar pruebas y generar informes. La integración continua suele ir asociada a las técnicas de programación extrema y desarrollo ágil.

Los pasos son:

1. Los desarrolladores envían sus modificaciones al Sistema de Control de Versiones.

2. Al ocurrir cambios en el repositorio, el servidor de integración continua ejecuta automáticamente la construcción.

3. Una vez finalizado el proceso de construcción (integración, construcción, pruebas y despliegue) el servidor notifica a los responsables con el resultado del proceso (retroalimentación del proyecto).

4. El servidor realiza el paso 2 continuamente.

Las diversas herramientas de integración continua permiten el despliegue automatizado del código en diversos servidores (servidores de desarrollo, servidores de prueba)

Principales beneficios: reducción del tiempo de integración, detección de errores los más pronto posible, pruebas inmediatas tras un cambio en el código, disponibilidad continua de la última versión del código para ejecutar pruebas y hacer demostraciones, así como disminución de los tiempos de desarrollo. (José Antonio Calvo Fernández 2007)

Impacto de la Integración Continua en cada fase del proceso:

En el ensamble: Generación automática de versiones, integración silenciosa, detección automática de errores e integración continua de versiones.

En pruebas: Detección rápida y temprana de errores, pruebas automáticas de regresión, verificación del cubrimiento de las pruebas unitarias, integración con herramientas de pruebas (unitarias, de carga, stress, funcionales) y trazabilidad con sistemas de seguimiento de errores.

En control de calidad: Monitoreo de estándares de codificación, monitoreo de malas prácticas y monitoreo de métricas de calidad.

En despliegue: Automatización de despliegue de aplicaciones en ambiente de desarrollo y pruebas.

1.2.1. Herramientas para integración continua

En los últimos años la integración continua se ha convertido en una técnica principal en el proceso de desarrollo de software, de ahí que cada día se hace más necesario el uso de las herramientas que la propician, entre las más conocidas se encuentran:

Apache Continuum

Continuum es una potente herramienta de Integración Continua desarrollada por Apache, de código abierto e implementada en Java, un producto relativamente joven que aunque ha evidenciado un avance desde sus primeras versiones (las cuales no alcanzaban el nivel de calidad mínimo exigido a este tipo de

soluciones) presenta muchas posibilidades de mejorar. Sin embargo, las nuevas funcionalidades que han ido apareciendo en las últimas versiones, unida a su intuitiva consola de administración, la hacen una buena herramienta y una opción seria para implementar una infraestructura de desarrollo basada en una plataforma de integración continua.

Es un software estable y fiable como la mayoría del software desarrollado por el grupo Apache. Una buena prueba de estabilidad del software es que está siendo utilizado internamente por varios de los proyectos gestionados por la propia Apache. (Apache Continuum 2011)

Ventajas: Posee una potente interfaz web.

Inconvenientes: No aporta ayuda en línea sobre los diversos conceptos y elementos de la interfaz, inexistencia de un mecanismo de extensión de funcionalidad que permita la inyección de plugins.

Hudson

Hudson es una potente y famosa herramienta de código abierto de libre distribución para llevar a cabo tareas de integración continua implementado en Java. Proporciona una interfaz sencilla y muy completa donde, además, se puede observar la evolución de un proyecto, teniendo como indicadores los diferentes fallos en la compilación, pruebas, etc. También guarda información de repositorio y de los cambios realizados en el código (al menos durante las compilaciones) para que se pueda llevar a cabo un rastreo de los errores que se produzcan.

Ventajas: Fácil instalación y uso, cuenta con un sistema de extensión de fácil ampliación, permitiendo la instalación y creación de nuevos plugins de forma sencilla y presenta soporte para compilación distribuida basada en sistemas esclavos y maestros, para mejorar los tiempos de compilación y evitar la sobrecarga. (Hudson 2011)

Inconvenientes: La tecnología Java usada por la herramienta, es poco conocida en el centro por lo que su utilización en este software implicaría un mayor riesgo de desarrollo, pues sería necesario el empleo de tiempo y esfuerzo adicional en el aprendizaje del lenguaje.

Buildbot

Buildbot es un interesante sistema para automatizar el proceso de compilación y testeo implementado en Python y de código abierto, necesitado por muchos productos de software para validar los cambios en el código, haciendo automáticamente continuas construcciones y pruebas a la raíz del código situado en un SCV. Se inspira en Tinderbox[1] pero con una aproximación más amplia, y con una característica fundamental como es la de solo realizar dichas construcciones y pruebas al código cuando este cambia. Por tanto, resuelve el problema del excesivo consumo de ciclos de CPU. (Buildbot 2011)

Buildbot consiste de dos partes:

− Un objeto maestro: Se encarga de escuchar eventos en el SCV. Si encuentra alguna modificación en el código fuente, notifica a sus esclavos que están en línea y se encarga de decir cuál va a empezar y qué tareas va a realizar cada uno. Se localiza en un servidor accesible por todos los esclavos y es controlado por un administrador del sistema. El maestro es la unidad central de control de todo el sistema. Todas las notificaciones sobre las modificaciones en el código son enviadas a él, todas las decisiones de qué es lo que se tiene que construir están en él, toda la información resultante de ejecutar las tareas pasa por él antes de ser distribuida para su visión pública. El maestro dispone de un directorio donde guardar la información para ser distribuida y también para guardar un historial de la ejecución de un proyecto.

− Objetos esclavos: Vinculados al maestro mediante puertos TCP e identificados con un usuario y contraseña. Se encargan de ejecutar las tareas asignadas a ellos. Cada esclavo puede ser de una plataforma diferente. Las tareas en este entorno reciben el nombre de constructores y consisten en un conjunto de comandos. Un tipo de constructor puede ser por ejemplo actualizar código desde repositorio SCV + compilar + probar.

La información es distribuida en Buildbot de diferentes maneras. Utiliza la herramienta Twisted[2] para comunicación cliente-servidor, lo que permite utilizar muchos protocolos. Una manera (la principal) es la de visualizar los resultados en una página web. Otro modo, y que no deja de ser interesante, es enviar los datos a un canal IRC (Internet Relay Chat), donde también el administrador puede gestionar remotamente el maestro.

[1] Tinderbox: Herramienta web que se utiliza para comprobar si el código existente en un momento dado en el SCV compila correctamente en varias plataformas.

[2] Twisted: Framework de red para programación dirigida por eventos.

Ventajas: Permite conocer al culpable de romper la aplicación, observando las últimas actividades de subida en el repositorio de control de versiones. Solo empezará a construir el código en el caso de que haya actividad en el repositorio de SCV. Permite enviar los resultados por diferentes protocolos. Permite que la gestión del maestro pueda ser remota y mediante diferentes protocolos. Permite que los esclavos sean controlados remotamente. En la página web, cada ejecución de tarea tiene una barra de progreso. Permite saber con más o menos exactitud cuánto tiempo queda para acabar una cierta tarea.

Inconvenientes: No ofrece gráficas de estadísticas.(Sordo 2006)

Selección de la herramienta a utilizar:

Después del estudio realizado a las herramientas antes mencionadas, se puede concluir que todas se caracterizan por ser potentes. Sin embargo se elige para el desarrollo de la aplicación la herramienta Buildbot, debido a que está hecho en Python, es muy completa, tiene distintas formas de notificación, además se puede catalogar como más configurable en cuanto a la inclusión de extensiones, trabaja con muchos SCV y tiene una documentación que es muy buena como referencia.

1.3. Pruebas de software

Cuando aparecieron los primeros sistemas informáticos se incluyó, a nivel metódico e imprescindible, un hasta entonces, nuevo proceso en la confección de los mismos: el proceso de pruebas.

Se calcula que la fase o proceso de pruebas puede representar hasta más de la mitad del coste y tiempo de desarrollo de un programa. Requiere muchas veces un tiempo similar al de la programación, lo que obviamente provoca un alto costo económico aún cuando estos no involucran vidas humanas, puesto que en este último caso el costo suele ser bastante elevado, siendo esta etapa más cara que el propio desarrollo y diseño de los distintos programas que conforman el sistema.

Las pruebas es una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

Pressman en el libro “Ingeniería de Software un enfoque práctico” para definir los objetivos de las pruebas hace referencia a Glem Myers quien establece algunos atributos que pueden servir para definir los objetivos de las mismas:

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado anteriormente.

Es importante resaltar que no siempre es necesario tener una aplicación funcionando para realizarle las evaluaciones pertinentes y lograr así su mejor funcionamiento, sino que si se lleva a cabo un proceso de prueba bien diseñado, que tenga pronosticado probar en cada una de las etapas del desarrollo, se eliminan muchos de los errores que posteriormente provocarían grandes gastos económicos, de tiempo y de esfuerzo humano.

1.3.1. Métodos de prueba

Los Métodos de Prueba de Software tienen el objetivo de diseñar pruebas que descubran diferentes tipos de errores con menor tiempo y esfuerzo.

Los dos métodos de prueba más utilizados son el Método de **Prueba de caja blanca (CB)**, Transparente o de cristal y el **Método de prueba de caja negra (CN).** La diferencia entre ambos radica en el hecho de que con la CB es necesario conocer el código y estar bastante relacionado con él para saber exactamente cuál es la lógica interna de lo que se va a probar, sin embargo en la CN solo basta conocer las posibles entradas y salidas del programa.

Ambos métodos son importantes a la hora de probar un software por muy pequeño que sea, un método complementa al otro. Pueden realizarse juntos o separados en dependencia de las necesidades y las características de la organización, pero la unión de ambos es la que garantiza la rápida detección y corrección de los errores que contiene el software, y que no han sido identificados aún por los desarrolladores.

Caja blanca

A las pruebas de CB también se les conoce como pruebas estructurales o pruebas de caja transparente. Están encaminadas a comprobar que las operaciones internas del programa se ajusten a las especificaciones y garantiza que todos los componentes internos se prueben adecuadamente.

Prueba de Caja Blanca: "Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba." (Pressman 1998)

El objetivo principal de estas pruebas consiste en garantizar que se ejerciten, por lo menos una vez, todos los caminos independientes de cada uno de los módulos, todas las decisiones lógicas en sus ambas variantes (verdadero y falso), todos los bucles en sus límites y con sus límites operacionales, y finalmente las estructuras internas de los datos para comprobar su validez. La prueba de CB del software se basa en el minucioso examen de los detalles procedimentales.

Caja negra

Las pruebas de CN, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software. O sea, la prueba de CN permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de CN no es una alternativa a las técnicas de prueba de CB. Más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de CB.(Pressman 1998)

Con este método, los casos de prueba y los resultados se determinan a partir de la especificación funcional de los métodos de una clase. Es decir, la prueba de CN se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Una prueba de CN examina algunos aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

1.3.2. Niveles y tipos de pruebas

Una vez que se ha generado el código comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de

errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos. (Galiano 2005)

El tamaño y complejidad del proceso de pruebas depende en gran medida del tamaño y complejidad del producto que se está desarrollando. Con el objetivo de simplificar el proceso y orientarlo a cada etapa del desarrollo del software, las pruebas se han dividido en varios niveles.

Pruebas unitarias

Cuando se implementa software, resulta recomendable comprobar que el código que hemos escrito funciona correctamente. Para ello se implementan pruebas que verifican que el programa genera los resultados esperados. Conforme se le añaden nuevas funcionalidades a las aplicaciones, se crean nuevas pruebas con los que medir el progreso y comprobar que lo que antes funcionaba sigue funcionando (prueba de regresión). Las pruebas de unidad también son de vital importancia: aunque no se añadan nuevas funcionalidades, se modifica la estructura interna del programa y se debe comprobar la no introducción de errores. Las pruebas de unidad son, por tanto, muy importantes en el desarrollo de software. Para agilizar el proceso resulta recomendable que una prueba sea completamente automática y compruebe los resultados esperados. (Galiano 2005)

Inicialmente, la prueba se centra en cada módulo de forma individual, asegurando que funcionan adecuadamente como una unidad. Dicha prueba hace un uso intensivo de las técnicas de prueba de CB, ejercitando caminos específicos de la estructura de control del módulo para asegurar un alcance completo y una detección máxima de errores. A continuación, se deben ensamblar o integrar los módulos para formar el paquete de software completo. (Pressman 1998)

Es importante tener en cuenta que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Además puede no ser trivial anticipar todos los casos especiales de entradas que puede recibir en realidad la unidad de programa bajo estudio. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

Pruebas de integración

La prueba de Integración se dirige a todos los aspectos asociados con el doble problema de verificación y de construcción del programa. Durante la integración, las técnicas que más prevalecen son las de diseño de casos de prueba de CN, aunque se pueden llevar a cabo algunas pruebas de CB con el fin de asegurar que se cubren los principales caminos de control. (Galiano 2005)

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. (Pressman 1998)

Se comprueba la compatibilidad y funcionalidad de las interfaces entre las distintas partes que componen un sistema, estas partes pueden ser módulos, aplicaciones individuales, aplicaciones cliente-servidor, etc. Este tipo de pruebas es especialmente relevante en aplicaciones distribuidas. (Prado 2007)

Estas pruebas tienen por objetivo verificar el funcionamiento de dos o más módulos. Se deben poner en práctica desde la creación de dos módulos que interactúen entre sí.

Es posible plantear las pruebas de integración desde dos puntos de vista: estructural o funcional:

- Las pruebas estructurales de integración son similares a las pruebas de CB; pero trabajan a un nivel conceptual superior. En lugar de referirse a sentencias del lenguaje, se refrieren a llamadas entre módulos. Se tratan de identificar todos los posibles esquemas de llamadas y ejercitarlos para lograr una buena cobertura de segmentos o de ramas.

- Las pruebas funcionales de integración son similares a las pruebas de CN. Se trata de encontrar fallos en la respuesta de un módulo cuando su operación depende de los servicios prestados por otro(s) módulo(s). Según se van acercando al sistema total, estas pruebas se van basando más y más en la especificación de requisitos del usuario.

Las pruebas finales de integración cubren todo el sistema y pretenden cubrir plenamente la especificación de requisitos del usuario.

Pruebas de sistema

Las Pruebas de Sistema verifican que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. Está constituida por varios tipos de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema. Algunas de estos tipos de pruebas son:

Prueba de validación: Proporciona una seguridad final de que el software satisface los requerimientos funcionales y de rendimiento.

Prueba de recuperación: Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.

Prueba de seguridad: Se determinan los niveles de permisos de usuarios, las operaciones de acceso al sistema y el acceso a datos.

Prueba de resistencia: Determinan hasta donde puede soportar el programa determinadas condiciones extremas.

Prueba de Rendimiento: Determinan los tiempos de respuesta, el espacio que ocupa el módulo en disco o en memoria, el flujo de datos que genera a través de un canal de comunicaciones, etc.

Prueba de Robustez: Determinan la capacidad del programa para soportar entradas incorrectas.

Prueba de Usabilidad: Se determina la calidad de la experiencia de un usuario en la forma en la que éste interactúa con el sistema, se considera la facilidad de uso y el grado de satisfacción del usuario.

Prueba de instalación: Se centra en comprobar que el sistema desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepcionales, por ejemplo con espacio de disco insuficiente o con continuas interrupciones.

Pruebas de aceptación

Son las que hará el cliente, se determina que el sistema cumple con lo deseado y se obtiene la conformidad del cliente. (Prado 2007)

Las pruebas de aceptación son realizadas principalmente por los usuarios, con el apoyo del equipo de desarrollo. El propósito es confirmar que el sistema está terminado, que desarrolla puntualmente las necesidades de la organización y que es aceptado por los usuarios.

Estas son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual de usuario. No se realizan durante el desarrollo, pues sería impresentable al cliente; sino que se realizan sobre el producto terminado y entregado, o pudiera ser una versión del producto o una iteración funcional pactada previamente con el cliente.

Una prueba de aceptación puede ir desde un informal caso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas. Estas pruebas son muy importantes, ya que definen el paso a nuevas fases del proyecto como el despliegue y mantenimiento. (Pressman 1998)

Este nivel de prueba es el último que se le aplica al software, si hasta el momento se ha llevado a cabo el proceso de pruebas cabalmente y como debe ser, este nivel no debe presentar dificultades demasiado grandes para los desarrolladores, en caso contrario existe la posibilidad de que el producto no pueda ser liberado y haya que corregir los errores o inconformidades encontrados a última hora.

1.3.3. Automatización de pruebas de software

El uso de software para probar los productos de software no es una idea novedosa. Sin embargo, es acogida por la mayoría de las empresas debido a su proceso acelerado y la aplicación más económica. Con el uso de software automatizado, en la fase de pruebas todo se ve muy reducido de semanas o meses a sólo días o incluso horas.

El término pruebas automatizadas se define como el proceso de automatización de determinadas pruebas que son originalmente llevadas a cabo de forma manual. Este proceso es parte de las estrategias utilizadas para disminuir el papel de humanos, especialmente en tareas que son redundantes.

Ventajas de la prueba automatizada

Confiable: Las pruebas realizan exactamente las mismas operaciones cada vez que se ejecutan, eliminando de tal modo los errores humanos.

Repetible: Es posible probar cómo reacciona el software bajo ejecución repetida de las mismas operaciones.

Reutilizable: Se puede reutilizar pruebas en diversas versiones de un producto.

Software con mayor calidad: Es posible realizar más pruebas en menos tiempo con pocos recursos.

Herramientas de automatización de pruebas

Según la metodología RUP las herramientas de prueba se pueden categorizar según las funciones que realicen. Algunas designaciones de funciones típicas para herramientas son:

Herramientas de adquisición de datos: Adquieren datos para utilizar en las tareas de prueba. Los datos se pueden adquirir mediante la conversión, la extracción, la transformación o la captura de datos existentes, o mediante la generación de guiones de uso o especificaciones suplementarias.

Herramientas estáticas de medida: Analizan información contenida en los modelos de diseño, el código fuente u otros orígenes fijos. El análisis produce información en el flujo lógico, el flujo de datos o la métrica de calidad, como la complejidad, el mantenimiento o las líneas de código.

Herramientas dinámicas de medida: Realizan un análisis durante la ejecución del código. Las medidas incluyen la operación de tiempo de ejecución del código, como la memoria, la detección de errores y el rendimiento.

Simuladores o controladores: Realizan tareas que, por cuestiones de tiempo, gastos o seguridad no están disponibles para las pruebas.

Herramientas de gestión de pruebas: Ayudan en la planificación, el diseño, la implementación, la ejecución, la evaluación y la gestión de tareas de prueba o productos de trabajo.

Además las herramientas de prueba suelen caracterizarse como cajas blancas o cajas negras en función de cómo se utilicen, o la tecnología y los conocimientos necesarios para utilizarlas.

Las herramientas de caja blanca dependen del conocimiento del código, los modelos de diseño y otro material de origen para implementar y ejecutar las pruebas.

Las herramientas de caja negra dependen de los guiones de uso o la descripción funcional del destino de la prueba.

Las herramientas de caja blanca saben cómo procesa la solicitud el destino de la prueba, mientras que las herramientas de caja negra dependen de las condiciones de entrada y de salida para evaluar la prueba.

A partir de la investigación realizada sobre los diferentes tipo de pruebas, serán seleccionadas para integrar a la solución, herramientas que apliquen fundamentalmente el método de prueba de CB, pues las mismas trabajan con el código fuente, además tienen la ventaja de que no necesitan tener un programa en ejecución para realizarle las pruebas, sino que desde la etapa de implementación los mismos desarrolladores pueden ejecutarlas, para asegurar la calidad del producto una vez terminado. Cabe destacar que en la selección de herramientas de CB tuvo un gran peso el hecho de que en el CEDIN hoy la mayor parte de las pruebas se realicen a las funcionalidades del software y no a su estructura interna, por lo que esta herramienta constituye una opción más para garantizar la calidad del producto.

1.4. Métricas de software

Todas las organizaciones de software exitosas implementan mediciones como parte de sus actividades cotidianas, pues estas brindan la información objetiva necesaria para la toma de decisiones y que tendrá un impacto efectivo en el negocio y desempeño en la ingeniería. Para poder asegurar que un proceso o sus productos resultantes son de calidad o poder compararlos, es necesario asignar valores, descriptores, indicadores o algún otro mecanismo mediante el cual se pueda llevar a cabo dicha comparación. (García 2008)

Para entender mejor el concepto de métrica es necesario aclarar que los términos, métricas, medición y medida no tienen el mismo significado.

Medida: Proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto. (Pressman 1998)

Medición: La medición es el acto de determinar una medida. (Pressman 1998)

Métrica: Es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. (Pressman 1998)

Se definen las métricas de software como la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos, para suministrar información relevante a tiempo, así el administrador junto con el empleo de estas técnicas mejorará el proceso y sus productos. (Doria 2001)

1.4.1. Clasificación de las métricas

Existen innumerables métricas con propósitos diferentes que reflejan o describen la conducta del software, estas pueden medir entre otros aspectos la competencia, calidad, desempeño y la complejidad del software contribuyendo a establecer de una manera sistemática y objetiva una visión interna del trabajo mejorando así la calidad del producto. A continuación se muestra la clasificación de las mismas: (Doria 2001)

Métricas de complejidad: Son todas las métricas de software que definen de una u otra forma la medición de la complejidad; tales como volumen, tamaño, anidaciones, costo (estimación) y configuración. Estas son los puntos críticos de la concepción, viabilidad, análisis, y diseño de software.

Métricas de competencia: Son todas las métricas que intentan valorar o medir las actividades de productividad de los programadores o practicantes con respecto a su certeza, rapidez, eficiencia y competencia.

Métricas de desempeño: Corresponden a las métricas que miden la conducta de módulos y sistemas de un software, bajo la supervisión del sistema operativo o hardware. Generalmente tienen que ver con la eficiencia de ejecución, tiempo, almacenamiento, complejidad de algoritmos computacionales, etc.

Métricas estilizadas: Son las métricas de experimentación y de preferencia; por ejemplo: estilo de código, las limitaciones, etc. Pero estas no se deben confundir con las métricas de calidad o complejidad.

Métricas de calidad: Son todas las métricas de software que definen de una u otra forma la calidad del software; tales como exactitud, estructuración o modularidad, pruebas, mantenimiento, reusabilidad, entre otras. Estas son los puntos críticos en el diseño, codificación, pruebas y mantenimiento.

La norma ISO 9126-1 establece métricas internas, externas, de calidad de uso y métricas para facilitar su evolución, que nos permiten medir las características de calidad del software, durante la producción y el desarrollo de las pruebas así como en la etapa de mantención.

Métricas internas son aquellas que no dependen de la ejecución del software (medidas estáticas) miden el comportamiento del sistema a partir del análisis del código por lo que son usadas principalmente en la etapa de desarrollo.

Métricas internas: Aplican a un producto de software no ejecutable durante las etapas de desarrollo, permiten medir la calidad de los entregables intermedios, predecir la calidad del producto final y al usuario, iniciar acciones correctivas temprano en el ciclo de desarrollo.

Ejemplos de Métricas Internas: Trazabilidad, número ciclomático, complejidad del flujo de información, tamaño del programa, enunciados condicionales, referencia unificada de datos, tamaño promedio de módulo (Mendoza 2006)

Las Métricas externas son aquellas que realizan las mediciones en función del comportamiento del sistema durante las pruebas y la operación del componente por lo que son usadas fundamentalmente en la etapa de pruebas.

Métricas Externas: Madurez, tolerancia a fallos, recuperabilidad, fiabilidad, precisión, estabilidad

Conociendo los diferentes tipos de métricas existentes, serán seleccionadas herramientas que apliquen métricas internas, ya que las mismas se comprueban con el código fuente y fundamentalmente en la etapa de desarrollo, lo que garantiza que al culminar este período, el producto contendrá mayor calidad gracias a la aplicación de estas métricas de calidad.

1.5. Metodología de desarrollo de software

Todo proceso de desarrollo de software para que se realice de manera exitosa debe estar basado en una metodología. No existe un proceso de desarrollo universal aplicable a todo proyecto. Factores como las características del equipo de desarrollo, el dominio de aplicación, el tipo de contrato, la complejidad y envergadura del proyecto, hacen necesario que las metodologías sean lo suficientemente adaptables como para poder aplicarse en distintos proyectos, y lo suficientemente sencilla para que no resulte muy incómoda su utilización, pero a la vez suficientemente completa como para que su uso por parte de un equipo de desarrollo sea provechoso. (Letelier 2008)

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo de software, con el objetivo de conseguir un software más eficiente y predecible (Cáceres y Marcos 2001). Aunque resultan poco adaptables, especialmente cuando se trata de proyectos pequeños. Mientras que las metodologías ágiles son consideradas más adaptativas que predictivas, teniendo como principales características la generación de pocos artefactos,modelado es prescindible, modelos desechables, pocos roles, más genéricos y flexibles, no existe un contrato tradicional, debe ser bastante flexible, cliente es parte del equipo de desarrollo, orientada a proyectos pequeños de corta duración y con menos de 10 integrantes. (Letelier 2008)

1.5.1. Proceso Unificado de Desarrollo de Software

Es una metodología de desarrollo pesada para la ingeniería de software que va más allá del análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software.

El Proceso Unificado de Desarrollo de Software (RUP) se caracteriza por ser iterativo e incremental donde cada fase se desarrolla en iteraciones, de forma tal que se pueda dividir en pequeños proyectos mejorando su comprensión y desarrollo. Esta metodología se centra en la arquitectura, aquí los modelos son proyecciones del análisis y el diseño constituye la arquitectura del producto a desarrollar.

Como RUP es un proceso, en su modelación define como sus principales elementos:

Trabajadores (quién),

Artefactos (qué).

Actividades (cómo).

Flujo de actividades (cuándo).

Un proyecto que se desarrolle utilizando la metodología RUP está dividido en 4 fases, 6 flujos de ingeniería y 3 de apoyo.

Fases: Inicio, Elaboración, Construcción, Transición.

Flujos de ingeniería: Modelo de negocio, Requisitos, Análisis y diseño, Implementación, Prueba, Despliegue.

Flujos de apoyo: Gestión de cambios y configuraciones, Gestión de proyectos, Ambiente o entorno.(KRUCHTEN 2000)

1.5.2. Desarrollo Guiado por la Funcionalidad

Desarrollo Guiado por la Funcionalidad (FDD) es una metodología ágil orientada a la funcionalidad del software por sobre las tareas, sobre las cuales da guías mínimas. No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción partiendo de una lista de características que debe reunir el software; provee estrategias de planeamiento para el líder de proyecto; fomenta la colaboración mediante la creación de equipos dirigidos por un programador jefe.

Un proyecto que sigue FDD se divide en 5 fases: Desarrollo de un modelo general, Construcción de la lista de funcionalidades, Plan de emisiones en base a las funcionalidades a implementar, Diseñar en base a las funcionalidades, Implementar en base a las funcionalidades. (Molpeceres 2002)

1.5.3. Programación Extrema

Programación Extrema (XP) es una metodología ligera utilizada para proyectos de corto plazo y equipos pequeños. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. Se define como especialmente adecuada para proyectos con funcionalidades imprecisas y muy cambiantes. Mientras que RUP intenta reducir la complejidad del software por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual, XP, como toda metodología

ágil, lo intenta por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción. (Molpeceres 2002)

La base para el desarrollo del software que usa esta metodología son las llamadas historia de usuarios, historias escritas por el cliente en las que describe escenarios sobre el funcionamiento del sistema y que no sólo están limitados a la interfaz de usuario, sino que también pueden describir modelos o dominios. Estas historias de usuarios junto a la arquitectura que se persigue, sirve de base para crear un plan de entregas de software entre el equipo de desarrollo y el cliente, para cada una de las cuales se definen los objetivos y las iteraciones (generalmente cortas) necesarias para cumplirlos (Molpeceres 2002). Las historias de usuarios y los casos de pruebas son la base sobre la que se asienta el trabajo del desarrollador.(Beck 1999)

XP impone un alto nivel de disciplina entre los programadores. Permite mantener un mínimo nivel de documentación, lo cual a su vez se traduce en una gran velocidad en el desarrollo (Hernán 2004)

A continuación se presenta las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles y proceso.

Las Historias de Usuario

Las historias de usuario son la técnica utilizada en XP para especificar las necesidades del software. Se trata de tarjetas en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

Respecto a la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. En muchos casos sólo se propone utilizar un nombre y una descripción. Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración. (Letelier 2008)

Roles de XP

Para una mejor organización de la ejecución de las actividades para el desarrollo de proyectos guiados por esta metodología se establecen los siguientes roles: Programador, Cliente, Encargado de pruebas, Encargado de seguimiento, Entrenador, Consultor, Gestor. (Letelier 2008)

Proceso XP

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1. (Letelier 2008)

El ciclo de vida ideal de XP consiste de 6 fases: Exploración, Planificación de entrega, Iteraciones, Producción, Mantenimiento, Muerte del Proyecto. (Letelier 2008)

Selección de la metodología de desarrollo

Todas las metodologías de desarrollo de software existentes presentan cualidades destacables como es el caso de la sencillez, tanto en el aprendizaje como en la aplicación a un proyecto de software determinado, permitiendo de esta forma reducir los costos de implantación en un equipo de desarrollo. Sin embargo, se decidió escoger una metodología ágil para el desarrollo de la propuesta, de ellas específicamente la metodología XP que es una metodología ligera utilizada para proyectos de corto plazo y equipos pequeños. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto, es importante destacar que en la selección de una metodología ágil influye el hecho de que la herramienta a desarrollar se realizará por solo una persona en un período corto de tiempo, proceso que se

haría engorroso con una metodología tradicional. Dada la novedad del tema en cuestión en el CEDIN, las necesidades actuales pueden estar sujetas a futuros cambios y éste es un punto en el cual esta metodología es flexible ya que permite administrar estos cambios de forma óptima.

1.6. Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software orientado a objetos (OO). Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.(Lafuente 2002)

UML no es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

1.7. Herramientas y lenguajes de desarrollo

Se propone utilizar tecnología Web para resolver el problema argumentado anteriormente, debido a que esta tecnología posee las siguientes ventajas:

- Se necesita solamente un navegador Web y conexión al sistema.
- Se pueden ejecutar varios clientes simultáneamente y en diferentes estaciones de trabajo.
- Por lo general no depende de ningún software.
- Generalmente no existe discriminación del sistema operativo por parte del usuario, aunque exista por parte del servidor.
- Se necesitan pocos recursos para que la aplicación trabaje correctamente.

1.7.1. Lenguajes de programación

El sistema que se implementará será una aplicación Web, por tanto, usará **arquitectura cliente – servidor**. En este caso el cliente solo estará relacionado con la interfaz externa del servidor, no con su lógica interna. Además, el cliente puede trabajar sin importar donde se encuentre el servidor, ni qué sistema operativo tenga, solamente necesita conexión hacia él. El servidor puede estar sujeto a cambios que deben influir poco o en nada en el cliente, también puede estar en diferente plataforma a la del servidor que no afectará el funcionamiento del sistema. Algo importante es que el servidor puede brindar servicios múltiples y simultáneos a clientes en cualquier lugar. Para llevar a cabo la implementación del sistema es necesario saber cual lenguaje de programación usar.

Para desarrollar un sistema Web los lenguajes de programación se pueden dividir en:

- Lenguaje del lado del cliente
- Lenguaje del lado del servidor

Dentro de este último caso (lenguaje del lado del servidor), se pueden apreciar lenguajes como PHP, ASP, JSP, Java. Dentro de los lenguajes del lado del cliente se encuentran HTML, CSS, JavaScript.

Preprocesador de Hipertexto (Hypertext Preprocessor, PHP)

PHP es un lenguaje de programación interpretado del lado del servidor, diseñado originalmente para la creación de páginas Web (Hinostroza 2005), gratuito, rápido, con una gran librería de funciones, mucha documentación y fácil de aprender. Este lenguaje es usado generalmente para el desarrollo de sitios Web, además presenta un estilo clásico, es decir, está contenido por bucles, variables, sentencias condicionales, funciones, entre otros. Este lenguaje de programación no requiere de definición de tipos de variables, permite las técnicas de Programación Orientada a Objetos y tiene manejo de excepciones desde PHP5. Es un lenguaje multiplataforma que disfruta de una amplia documentación en su página oficial. Puede usarse con la mayoría de los sistemas operativos, ejemplo Windows o basados en Unix. Presenta una gran capacidad de conexión con una marcada cantidad de gestores de base de datos que son utilizados actualmente (ejemplo: PostgreSQL, Oracle, dbm, filePro, interbasem), aunque su mayor conectividad, la base de datos con la que mejor trabaja es con MySQL. Permite crear los formularios para la Web, así como las técnicas de Programación Orientada a Objetos. Es un lenguaje que posee muchas

ventajas a su favor; aunque en ocasiones pueden mezclarse las sentencias HTML y PHP por lo que se ve afectada la claridad del código.

Ventajas: Software libre, multiplataforma, fácil de utilizar, fácil de integrar con MySQL, permite crear los formularios, biblioteca nativa de funciones sumamente amplia e incluida, posee la capacidad de expandir su potencial utilizando una enorme cantidad de módulos (llamados extensiones) y no requiere definición de tipos de variables ni manejo detallado del bajo nivel.

Lenguaje de Marcado de Hipertexto (HyperText Markup Language, HTML)

HTML es el lenguaje de marcado predominante para la construcción de páginas web. Usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. Este lenguaje está hecho de etiquetas y atributos que trabajan conjuntamente para dar alguna característica específica a la página web; el navegador interpreta dichas etiquetas y atributos y las despliega en la pantalla, solo se limita a describir la estructura y el contenido de un documento y no el formato de la página y su apariencia.

Hojas de Estilo en Cascada (Cascading Style Sheets, CSS)

Es la tecnología desarrollada por el World Wide Web Consortium (W3C) con el fin de separar la estructura de la presentación. A pesar de que la recomendación oficial del grupo de trabajo de la W3C ya había alcanzado la estabilidad requerida para que fuera soportada por los principales navegadores comerciales, como Netscape e Internet Explorer. (Álvarez 2000)

La potencia de la tecnología CSS permite aplicar al documento formato de modo mucho más exacto, tiene como principales ventajas que permite definir la distancia entre líneas del documento, colocar elementos en la página con mayor precisión y sin dar lugar a errores y definir la visibilidad de los elementos, márgenes, subrayados y tachados.

Java Script

Es un lenguaje de scripts desarrollado por Netscape[3] para incrementar las funcionalidades del lenguaje HTML. Se utiliza embebido en el código HTML y XHTML, entre las etiquetas <script> y </script>. Dentro de sus características más importantes se pueden encontrar:

- Es un lenguaje interpretado, es decir, no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias Java Script contenidas en una página HTML y ejecutarlas adecuadamente.

- Es un lenguaje orientado a eventos. Cuando un usuario presiona el cursor sobre un enlace o mueve el puntero sobre una imagen se produce un evento. Mediante este lenguaje se pueden desarrollar scripts que ejecuten acciones en respuesta a estos eventos.

- Es un lenguaje orientado a objetos. El modelo de objetos de Java Script está reducido y simplificado, pero incluye los elementos necesarios para que los scripts puedan acceder a la información de una página y puedan actuar sobre la interfaz del navegador. (Pérez 2006)

1.7.2. Editor para desarrollo Web

Bluefish Editor es un potente editor dirigido a programadores y diseñadores web, con muchas opciones para escribir páginas web, scripts y código de programación, centrándose en la edición de sitios web dinámicos e interactivos. Soporta una multitud de lenguajes tales como HTML, JavaScript, CSS, PHP, Java, JSP, C, SQL, Python, entre otros.

Bluefish cuenta con características tales como rapidez, posibilidad de abrir múltiples archivos simultáneamente, diálogos para etiquetas HTML, asistentes para creación fácil de documentos, soporte para múltiples codificaciones, menús desplegables, barras de herramientas configurables, diálogo para insertar imágenes, buscador de referencia de funciones y resaltado de sintaxis. Es además un proyecto de desarrollo de código abierto, liberado bajo la licencia GPL de GNU. (Bluefish Editor 2011)

1.7.3. Herramienta CASE de Modelado con UML

[3] Netscape Communications Corporation: Empresa de desarrollo de software famosa por ser la creadora de del navegador web Netscape Navigator.

Visual Paradigm es una herramienta CASE que utiliza UML como lenguaje de modelado profesional la cual soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite realizar ingeniería tanto directa como inversa. La herramienta es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto.

Entre sus principales características se encuentran el uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación, el modelo y el código permanecen sincronizados en todo el ciclo de desarrollo, posee disponibilidad en múltiples plataformas, cuenta con una interfaz amigable, permite la generación de documentos, posee interoperabilidad con otras aplicaciones, permite la integración con distintos Ambientes de Desarrollo Integrados (IDE), y tiene facilidad de instalación y actualización. (Hernandis 2005)

1.7.4. Sistema Gestor de Bases de Datos (SGBD)

Un Sistema Gestor de Bases de Datos (SGBD) o DBMS (DataBase Management System) es un conjunto de programas que se encargan de manejar la creación y todos los accesos a las bases de datos. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Su función fundamental es proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos.

MySQL Server es la base de datos de código fuente abierto más usada del mundo desarrollado y proporcionado por MySQL AB. El servidor fue desarrollado originalmente para manejar grandes bases de datos mucho más rápido que las soluciones existentes y ha sido usado exitosamente en ambientes de producción sumamente exigentes por varios años.

Una de las principales ventajas de MySQL es que es multiplataforma, posee una estabilidad comprobada, gran rapidez a la hora de ejecutar las consultas, conectividad segura y una excelente integración con PHP.

Como sus trascendentales características podemos señalar que soporta hasta 32 índices por tabla, una gran cantidad de tipos de datos para las columnas, aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo, gran portabilidad entre sistemas. (Castellanos y Pabón 2005)

1.7.5. Sistema de Control de Versiones (SCV)

Los Sistema de Control de Versiones son sistemas que están diseñados para registrar y seguir los cambios en los datos a través del tiempo (Sussman, Fitzpatrick, y Pilato 2002) y cuya utilización es clave en el desarrollo de cualquier producto en el que interviene más de una persona. (Gayo et al. 2006)

Git es el sistema que actualmente se usa como control de versiones en el núcleo Linux, y que fue diseñado por el ingeniero Linus Torvalds. Entre las características más importantes, está su eficiencia para aplicaciones con muchos archivos fuentes, y también apuesta por un desarrollo distribuido con muchas ramificaciones de la misma aplicación. Esto también hace que este sistema tenga una complejidad importante a la hora de resolver conflictos, por su alto nivel de distribución y por su capacidad para desarrollar distintas ramas, que hace que las fusiones entre ellas para generar el programa final pueda ser más complicado.(Iglesias 2009)

Consideraciones parciales

En este capítulo quedó demostrado que el proceso de integración cuando se realiza de manera continua, tiene como resultado disímiles aportes al proceso de desarrollo del software y por lo tanto al producto final, por lo que se propone Buildbot para el desarrollo de la solución debido a que la misma es una potente herramienta que cuenta con una amplia documentación y facilidad de uso, además, conociendo que en un entorno de integración continua, las pruebas y las métricas son ejecutadas de manera regular para probar que el software desarrollado funciona correctamente con cada cambio realizado, la aplicación a desarrollar tiene que propiciar la integración (al entorno de integración continua) de herramientas de pruebas automatizadas, fundamentalmente de aquellas que realizan una evaluación directa del código y de otras que apliquen y/o evalúen métricas internas en un producto de software.

Se utilizarán las técnicas de recopilación de la información adecuadas como el uso del lenguaje unificado de modelado (UML), cuya utilización de diagramas y gráficos brinda una mejor perspectiva de lo que se quiere. Visual Paradigm es la herramienta de modelado a usar, ya que se caracteriza por ser intuitiva para usuarios que no poseen la costumbre de trabajar con UML. PHP es el lenguaje a utilizar pues es poderoso para desarrollar aplicaciones Web del lado del servidor.

CAPÍTULO 2
DESARROLLO DE LA SOLUCIÓN

Introducción

En el presente capítulo se describen brevemente las principales funciones y el flujo actual de los procesos involucrados en el problema en cuestión, haciéndose un análisis de cómo se ejecuta actualmente el proceso de integración y pruebas de código en el CEDIN.

Se muestran las historias de usuarios que fueron escritas por el cliente y se definen tres iteraciones por los programadores, elaborándose el Plan de Entrega para cada versión. Se definen las tareas generadas a partir del desarrollo de las historias de usuarios durante las tres iteraciones planificadas, se puntualizan los artefactos que dirigen la descripción del sistema y las pruebas de aceptación diseñadas.

2.1. Descripción de las acciones vinculadas al campo de acción

Actualmente la integración de código en el desarrollo de los productos del CEDIN se hace muy engorrosa. Esto se debe a que es necesaria la aplicación de largas horas de trabajo para integrar de forma manual el código fuente obtenido por diversos desarrolladores que implementan de manera simultánea; así como la corrección de errores producto al prolongado tiempo entre la integración de una versión del código y la siguiente. Errores que se producen además, debido a que antes de la integración, el código no se somete a pruebas que garanticen la calidad del mismo.

Lo antes planteado constituye un inconveniente para la obtención de resultados fiables y para la rapidez del proceso de desarrollo e integración.

2.1.1. Flujo actual de eventos

El proceso de desarrollo e integración en la actualidad está definido por una serie de actividades que fluyen de la siguiente forma:

Los desarrolladores implementan el código de un producto de software determinado de manera simultánea, una vez obtenido el mismo se procede a la integración de los diferentes componentes donde se corrigen los errores y se le realizan pruebas para comprobar el correcto funcionamiento. Este proceso ocurre de forma manual por lo que pueden perderse y omitirse datos importantes.

2.1.2. Objetos de automatización

Durante el proceso de desarrollo e integración existen varias acciones que deben ser automatizadas, puesto que se hace engorroso el manejo del código y se tiende a cometer a errores.

Se automatizarán las actividades de integración de código fuente propiciando un entorno de integración continua, con el objetivo de disminuir los tiempos de integración, así como la realización de pruebas y aplicación de métricas a código para obtener un producto de mayor calidad.

2.1.3. Propuesta del sistema

Con la realización del siguiente trabajo se dispone desarrollar la implementación de una herramienta que permita la gestión (insertar, modificar, eliminar y consultar) de los datos del código de los proyectos del CEDIN, necesarios para la integración así como la realización de pruebas automatizadas y aplicación de métricas de calidad al mismo, generando reportes de los resultados. El sistema permitirá además, la consulta de reportes realizados en construcciones anteriores, así como la descarga de estos.

Existirá un espacio dedicado a la publicación de la información sobre los módulos incluidos a la herramienta. Cada módulo será el responsable de las pruebas y métricas que se le apliquen al código.

Se habilitará el sistema de manera que pueda ser usado por cualquier usuario del dominio UCI, con un solo administrador para limitar el acceso a servicios brindados por la herramienta a quienes no lo requieran.

La solución debe cumplir como mínimo las siguientes condiciones:

La disponibilidad del sistema debe ser continua con un nivel de servicio para los usuarios de 24 horas toda la semana.

Autenticación de usuarios: Todos los usuarios que requieran acceder al sistema, deben ser identificados y autenticados contra un repositorio o Base de Datos única de identidades (dominio UCI), acción que permitirá habilitar los permisos que tendrán los mismos al ingresar al sistema.

La herramienta deberá poseer la capacidad de ser extensible en cuanto a los módulos de prueba se refiere.

2.2. Exploración

Como la metodología XP propone, en esta fase los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo, el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. (Letelier 2008)

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La extensión de la fase de exploración se sugiere que sea de 2 ó 3 semanas hasta pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Para el desarrollo de la herramienta que se desea implementar solo se utilizarán 2 semanas, debido a que el mismo no es un sistema de gran extensión y el equipo de desarrollo tiene dominio del trabajo con las herramientas que se utilizarán.

2.2.1. Actores del sistema

Actores	Descripción
Usuario	Hace uso de la aplicación, beneficiándose de sus funcionalidades.
Administrador	Es una especialización de usuario, hace uso de la herramienta, con privilegios en la Gestión de proyectos que permiten el mantenimiento.
Usuario Restringido	Es una especialización de usuario, hace uso de la herramienta, realizando la Gestión de proyectos con privilegios restringidos.
Buildbot	Agente externo que realiza la integración continua del código.

Sistema	Ejecuta los módulos de prueba de manera automática

Tabla 1. Actores del sistema

2.2.2. Historias de usuario

En XP la gestión de requerimientos del sistema es extremadamente simple, el cliente describe y prioriza sus necesidades mediante historias de usuario (HU). Son descripciones cortas y escritas sin terminología técnica.

Las historias de usuario solamente proporcionaran los detalles sobre la complejidad y cuánto tiempo conllevará la implementación de dicha historia de usuario. El nivel de detalle de las historias de usuario debe ser el mínimo posible que permita hacerse una ligera idea de cuánto costará implementar el sistema.

Los programadores estiman el esfuerzo asociado y las dependencias entre ellas. Para planificar el trabajo desde el punto de vista técnico, las HU son divididas en tareas para las cuales también se realiza una estimación. Teniendo en cuenta el esfuerzo asociado a las HU y las prioridades del cliente se define una versión que sea de valor y que tenga una duración de unos pocos meses.

Historia de Usuario	
Número: 1	**Nombre Historia de Usuario:** Insertar Proyecto
Actor: Usuario	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 0.5
Nivel de Complejidad: Media	**Puntos Reales:** 0.5
Descripción: El sistema debe permitir al usuario insertar un proyecto. Los datos asociados son: Nombre, SCM, Url, Rama, Directorio del código fuente, Lenguaje, Herramienta de construcción, Planificadores.	
Observaciones:	

Tabla 2. Insertar Proyecto

Historia de Usuario	
Número: 2	**Nombre Historia de Usuario:** Consultar Proyecto

Actor: Usuario	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 0.5
Nivel de Complejidad: Media	**Puntos Reales:** 0.5
Descripción: El sistema debe permitir al usuario consultar los proyectos existentes en la herramienta.	
Observaciones:	

Tabla 3. Consultar Proyecto

Historia de Usuario	
Número: 3	**Nombre Historia de Usuario:** Gestionar Proyecto
Actor: Usuario Restringido	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Nivel de Complejidad: Media	**Puntos Reales:** 1
Descripción: El usuario restringido podrá modificar o eliminar solo los proyectos creados por él. Los datos asociados son: Nombre, SCM, Url, Rama, Directorio del código fuente, Lenguaje, Herramienta de construcción, Planificadores.	
Observaciones:	

Tabla 4. Gestionar Proyecto

Historia de Usuario	
Número: 4	**Nombre Historia de Usuario:** Gestionar Grupos de Proyectos
Actor: Administrador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Nivel de Complejidad: Media	**Puntos Reales:** 1
Descripción: El administrador podrá modificar o eliminar todos los proyectos existentes en la herramienta. Los datos asociados son: Nombre, SCM, Url, Rama, Directorio del código fuente, Lenguaje, Herramienta de construcción, Planificadores.	
Observaciones:	

Tabla 5. Gestionar Grupos de Proyectos

Historia de Usuario

Número: 5	**Nombre Historia de Usuario:** Autenticar Usuario
Actor: Usuario	**Iteración Asignada:** 2
Prioridad de Negocio: Alta	**Puntos Estimados:** 1
Nivel de Complejidad: Baja	**Puntos Reales:** 1
Descripción: El sistema debe permitir el acceso a los usuarios mediante la autenticación en el dominio UCI, habilitándole los permisos asignados para el usuario.	
Observaciones:	

Tabla 6. HU Autenticar Usuario

Historia de Usuario	
Número: 6	**Nombre Historia de Usuario:** Realizar integración
Actor: Buildbot	**Iteración Asignada:** 2
Nivel de Complejidad: Alta	**Puntos Estimados:** 2
Nivel de Complejidad: Alta	**Puntos Reales:** 2
Descripción: Buildbot debe realizar el proceso de integración de código cada cierto tiempo, generando un reporte con el resultado de la misma.	
Observaciones:	

Tabla 7. HU Realizar Integración

Historia de Usuario	
Número: 7	**Nombre Historia de Usuario:** Aplicar métricas de calidad y pruebas a código
Actor: Sistema	**Iteración Asignada:** 3
Prioridad de Negocio: Alta	**Puntos Estimados:** 2
Nivel de Complejidad: Alta	**Puntos Reales:** 2
Descripción: El sistema debe realizar las pruebas y aplicar métricas de calidad al código existente cada cierto tiempo, generando un reporte con el resultado.	
Observaciones:	

Tabla 8. HU Aplicar métricas de calidad y pruebas a códigos

Historia de Usuario

Número: 8	**Nombre Historia de Usuario:** Consultar reporte
Actor: Usuario	**Iteración Asignada:** 3
Prioridad de Negocio: Alta	**Puntos Estimados:** 0.5
Nivel de Complejidad: Baja	**Puntos Reales:** 0.5
Descripción: El sistema debe permitir al usuario consultar los reportes realizados anteriormente.	
Observaciones:	

Tabla 9. HU Consultar reporte

Historia de Usuario	
Número: 9	**Nombre Historia de Usuario:** Descargar reporte
Actor: Usuario	**Iteración Asignada:** 3
Prioridad de Negocio: Media	**Puntos Estimados:** 0.3
Nivel de Complejidad: Baja	**Puntos Reales:** 0.3
Descripción: El sistema debe permitir al usuario descargar cada reporte consultado.	
Observaciones:	

Tabla 10. HU Descargar reporte

Historia de Usuario	
Número: 10	**Nombre Historia de Usuario:** Consultar información de módulos
Actor: Usuario	**Iteración Asignada:** 3
Prioridad de Negocio: Media	**Puntos Estimados:** 0.2
Nivel de Complejidad: Baja	**Puntos Reales:** 0.2
Descripción: El sistema debe permitir al usuario descargar cada reporte consultado.	
Observaciones:	

Tabla 11. HU Descargar reporte

2.2.3. Diseño de Casos de Pruebas

A diferencia las metodologías tradicionales, donde la fase de pruebas, incluyendo la definición de las mismas, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo; la metodología XP propone un modelo inverso, en el que, lo primero que se escribe son las pruebas que el sistema debe pasar. (Joskowicz, 2008)

Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las historias de usuarios definidas por el cliente. Las pruebas de aceptación son consideradas como pruebas de caja negra. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. (Joskowicz 2008)

Las pruebas de aceptación son creadas a partir de las HU. Durante una iteración la HU seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. El cliente o usuario especifica los aspectos a testear cuando una HU ha sido correctamente implementada. Una HU puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento. (Escribano 2002) Una historia de usuario no se puede considerar terminada hasta que no pase correctamente todas las pruebas de aceptación.

Para la realización de las pruebas a la herramienta se diseñaron 12 casos de pruebas donde se ejecutan paso a paso cada una de las posibles entradas al sistema y se evalúa el resultado de las mismas.

Las tablas con los diseños de casos de pruebas se encuentran en el epígrafe 3.1 del capítulo 3 donde además se le incluyen los resultados de la ejecución de las mismas.

2.3. Planificación y entrega

La planificación es una fase corta en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario y asociadas a éstas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas.(Joskowicz 2008)

En esta fase el cliente establece la prioridad de cada HU y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el

contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debe obtenerse en no más de tres meses.

2.3.1. Plan de Entrega

Una vez que el cliente culmina la elaboración de las HU, se comienza con la creación del Plan de Entregas. El mismo se hace con la intención de que los programadores obtengan una estimación de dichas historias en cuanto al nivel de detalle, o sea, para fijar el período de tiempo que se puede tardar en la implementación de cada una.

Historias de usuario	Iteración 1 (7 /3/2011)	Iteración 2 (21/3/2011)	Iteración 3 (11/3/2011)
Insertar Proyecto Consultar Proyecto Gestionar Proyecto Gestionar Grupos de Proyectos	3	Terminado	Terminado
Autenticar Usuario Realizar integración	No empezado	3	Terminando
Aplicar métricas de calidad y pruebas a código Consultar reporte Descargar reporte Consultar información de módulos	No empezado	No empezado	3

Tabla 12. Plan de entrega

Historias de usuario	Tiempo estimado (semana ideal de trabajo)	Iteración asignada	Tiempo real
Insertar Proyecto Consultar Proyecto Gestionar Proyecto Gestionar Grupos de Proyectos	3	1	3
Autenticar Usuario Realizar integración	3	2	3
Aplicar métricas de calidad y pruebas a código Consultar reporte Descargar reporte Consultar información de módulos	3	3	3

Tabla 13. Estimación del esfuerzo por historia de usuarios

A pesar de que la planificación realizada cumpla con todos los requerimientos, los riesgos son altos ya que el tiempo es corto y algunas de las historias de usuarios poseen una alta complejidad. Debido a estas circunstancias, se tomaron medidas que permitirán el desarrollo de la solución de forma segura y eficiente.

- Mantener un control de versiones del desarrollo de la aplicación, que permita regresar a una versión anterior y funcional si ocurriese algún problema.

- Comprobación periódica, que permita probar el funcionamiento correcto de la aplicación y evitar que algunas funcionalidades presenten errores debido a la incorporación otras o por el constante cambio en el código.

2.4. Diseño del sistema

La metodología XP no requiere la descripción del sistema por medio de diagramas de clase utilizando notación UML, sino que se guía por técnicas como las tarjetas CRC (Contenido, Responsabilidad y Colaboración). Esto no implica que no se utilicen los diagramas para obtener una mejor visión y comunicación entre el equipo de trabajo, siempre y cuando su complejidad no sea alta y defina información importante.

2.4.1. Tarjetas CRC

Las características más sobresalientes de las tarjetas CRC son su simpleza y ductilidad. Una tarjeta CRC no es más que una ficha de papel o cartón que representa a una entidad del sistema. (Casas y Reinaga 2008)

Estas tarjetas se utilizan para estructurar las clases y a su vez definir las responsabilidades sobre las mismas, así como la simulación de escenarios en el sistema.

Clase	
Responsabilidades	Colaboradores

Tabla 14. Modelo de Tarjeta CRC.

Tarjetas CRC del sistema

El sistema a desarrollar dispone de diversos objetos, por lo que se definen las siguientes clases:

Report, Project, Builder, Control, Extension, ObjectRepository, SherlockConf, DBManager, LibInitData

A continuación se definen las tarjetas CRC del sistema:

Extension	
Manejo de los módulos de prueba	ObjectRepository Project LibInitData

Tabla 15. Tarjeta CRC para la clase Extension

Builder	
Realizar comunicación con Buildbot	Report SherlockConf Project

Tabla 16. Tarjeta CRC para la clase Builder

Project	
Contenedora de datos del proyecto	

Tabla 17. Tarjeta CRC para la clase Project

Report	
Contenedora de datos del reporte	

Tabla 18. Tarjeta CRC para la clase Report

LibInitData	
Contenedora de datos de la extensión	

Tabla 19. Tarjeta CRC para la clase LibInitData

ObjectRepository

Manipula los objetos del repositorio	Extension

Tabla 20. Tarjeta CRC para la clase ObjectRepository

DBManager	
Ejecutar consultas en la Base de Datos	Report SherlockConf Project

Tabla 21. Tarjeta CRC para la clase DBManager

SherlockConf	
Almacena datos de configuración del Sherlock	

Tabla 22. Tarjeta CRC para la clase SherlockConf

Control	
Realizar comunicación con capa de acceso a datos.	DBManager Report Project

Tabla 23. Tarjeta CRC para la clase Control

2.4.2. Extensibilidad

La extensibilidad de un sistema determina su capacidad de extenderse y ser reimplementado en diversos aspectos, o sea, que puedan ser añadidos o eliminados componentes del mismo sin afectar su funcionamiento. En la actualidad, la integración de componentes escritos por diferentes programadores constituye un auténtico reto.

La herramienta a desarrollar poseerá la capacidad de ser extensible en cuanto a los módulos de prueba se refiere, deberá de ser implementada de manera tal que una vez terminada, sea posible la integración de nuevos módulos que apliquen determinadas pruebas añadiéndole valor a la aplicación. Módulos que no necesariamente tienen que ser implementados por la misma persona, aunque si debe cumplir con las características estructurales del software.

Un módulo es un paquete que contiene como mínimo un fichero module.php con una clase que presenta la siguiente estructura:

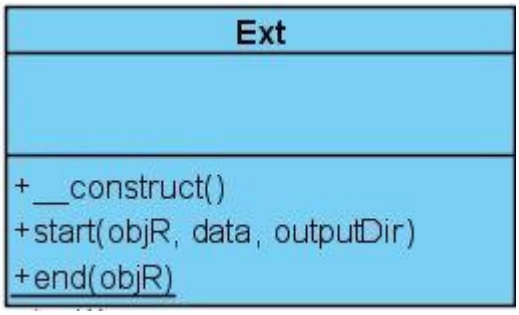

Figura 2. 1 Estructura de la clase de un módulo

Es la función start la que ejecuta las principales funcionalidades del módulo, siendo de vital importancia guardar el resultado de la ejecución del mismo en un fichero (result.html) en $outputDir.

La extensibilidad de la herramienta será probada inicialmente con la inserción de algunos módulos que realicen pruebas a código y verifiquen métricas de calidad al mismo.

Módulos a incluir en la Herramienta:

El módulo **Cflow**, permitirá el analizar una colección de archivos de código fuente escrito en lenguaje de programación C, mostrando un gráfico de dependencias entre las distintas funciones.

El módulo **Valgrind**, permitirá detectar problemas de memoria imprimiendo un reporte con los resultados.

El módulo **testApps** permitirá la generación de diversos reportes debido a la utilización de herramientas de pruebas y métricas, entre las que se encuentran:

CCCC - Herramienta que analiza archivos en C, C++ y Java y genera un informe con varias métricas del código. Entre las métricas apoyadas incluyen líneas de código, la Complejidad Ciclomática, entre otras.

SLOCCount - (del inglés cómputo de líneas de código fuente) Analiza proyectos de software, haciendo la estimación de coste, duración y número de desarrolladores. Cuenta las líneas físicas del código fuente,

aunque esté en diferentes lenguajes, basado en medios tradicionales en la ingeniería de software con el modelo COCOMO.

Flawfinder - Herramienta de Python diseñada para analizar código fuente C y C++ en busca de potenciales debilidades de seguridad, generando informes de posibles fallos ordenados por nivel de riesgo. Funciona utilizando una base de datos incorporada de C / C++ con problemas bien conocidos, como los riesgos de desbordamiento de búfer, problemas de formato de cadena, entre otros.

RATS – Herramienta que analiza código C, C++, Perl, PHP y Python y al finalizar muestra una lista con los potenciales problemas de seguridad, una descripción del problema y una posible solución para fortificar la aplicación. También proporciona un gravamen relativo de la severidad potencial de cada problema, para ayudar a priorizar los fallos de seguridad.

Estas dos últimas herramientas tienen funcionalidades muy parecidas, ambas reportan problemas de seguridad en el código, sin embargo se decidió unirlas pues una sirve de complemento a la otra, entre las diferencias que presentan se encuentran por ejemplo que RATS tiene soporte para lenguajes de programación con los que Flawfinder no cuenta, por su parte esta última puede manejar programas internacionalizados, reportar números de columna (así como los números de línea) de accesos, etc.

2.4.3. Diagrama de Despliegue

La metodología XP plantea que para un mejor entendimiento de las tareas, flujos y métodos de desarrollo de las funcionalidades se pueden crear diagramas, siempre que su creación no implique mayor esfuerzo que la implementación del mismo. Siguiendo este principio se elaboró el diagrama de despliegue que permite apreciar de forma visual cómo se encuentran relacionados físicamente los componentes en la aplicación.

Un diagrama de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre nodos de cómputo.(Jacobson, Booch, y Rumbaugh 2000). El sistema implementado se encuentra ubicado en un servidor Web Apache con conexión a una base de datos *MySQL*, el Servidor BuildMaster que controla la integración y los BuildSlave encargados de ejecutar la misma.

Estos servidores pueden encontrarse en diferentes nodos físicos, pues una de las características de los mismos es que permiten la comunicación a través de la red, sin embargo debido a las condiciones del centro para el que se realiza esta herramienta, todos los servicios se encontrarán en un mismo nodo físico.

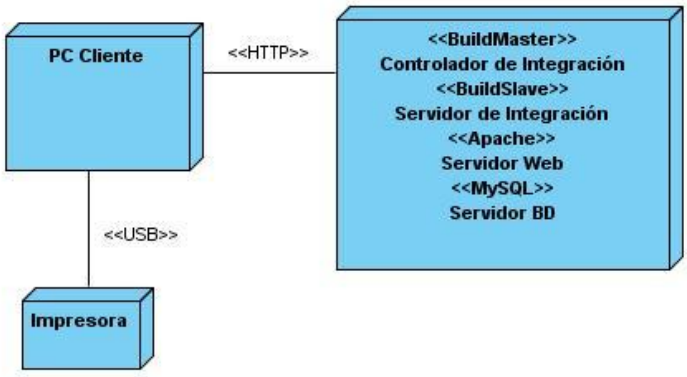

Figura 2. 2 Diagrama de Despliegue

2.4.4. Arquitectura del sistema

El diseño de la arquitectura de un sistema es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes conforman el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada.

Se hará uso de una arquitectura multicapas. Para este sistema la misma estará dividida en tres capas que tienen diferentes propósitos e interactúan entre sí pero de manera independiente, de forma tal que al hacer cambios en una no se afectan los otros dos , básicamente estos módulos o capas reciben el nombre de:

Capa de presentación: Es la capa encargada de hacer el pedido y mostrar la información al usuario, es donde va la interfaz de usuario.

Lógica de aplicaciones: Esta capa es la encargada de recibir el pedido del cliente, extrae los datos al interactuar con la capa de almacenamiento y da una respuesta al cliente, es donde van implementadas las reglas de negocio.

Almacenamiento: Almacena los datos que serán utilizados por la aplicación.(Torre et al. 2010)

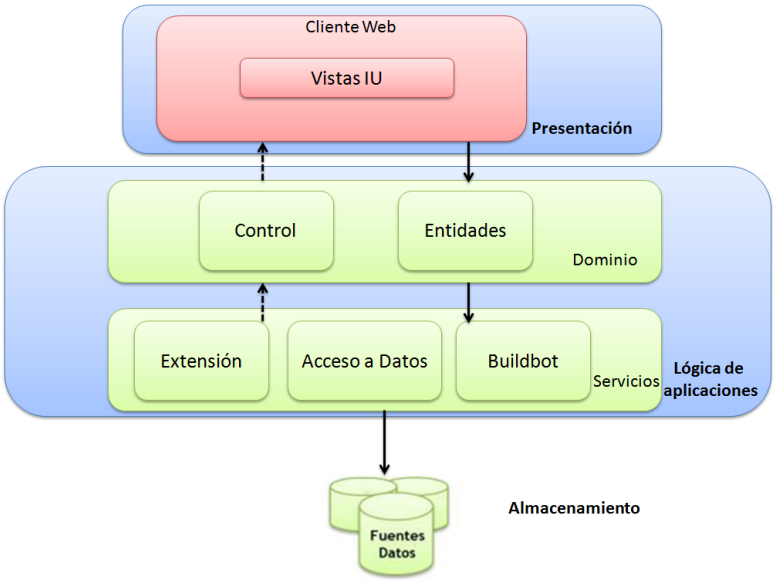

Figura 2. 3 Arquitectura del sistema

2.4.5. Patrones de diseño

Los Patrones son soluciones comunes a problemas de diseño de software orientado a objetos y que además poseen ciertas características de efectividad para resolver ese problema. Son reusables ya que pueden ser aplicados en otros diseños o problemas.(Rojas 2007)

Se hará el uso del patrón Singleton en las clases DBManager y ObjectRepository para garantizar que sólo exista una instancia de cada una, ya que el mismo está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

2.4.6. Estándares de codificación

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. (Carlos Fernández 2008) La metodología XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados). Los estándares de programación mantienen el código legible para los miembros del equipo facilitando los cambios. (Letelier 2008) Para la implementación del sistema desarrollado se siguieron normas y estándares desarrollados por el equipo, que se relacionan a continuación.

Definiciones generales

Las definiciones se realizan en inglés de manera descriptiva, evitando las abreviaturas y los nombres cortos

Clases:

Las clases comienzan con mayúscula al inicio de la palabra y en caso de estar conformada por palabras compuestas, la definición debe ser continua y cada palabra debe iniciar con mayúscula siguiendo el estilo determinado.

Ejemplos:

abstract class Extension {}

class ObjectRepository {}

Funciones:

Las funciones deben empezar con minúscula. En caso de estar conformados por palabras compuestas, para las que se encuentran dentro de clases, la definición debe ser continua y exceptuando la primera, cada palabra debe iniciar con mayúscula siguiendo el estilo determinado; para las que se encuentran fuera de clases deben estar separadas por guión bajo y cada palabra debe iniciar con minúscula siguiendo el estilo determinado.

Ejemplos:

public function start ($objR, ProjectData $data, $outputDir)

Dentro de la clase: public function getHolder()

Fuera de la clase: public function get_holder()

Variables:

Las variables comenzarán con minúsculas, aquellas que sean compuestas se escriben de manera seguida y con las primeras letras de cada palabra en mayúsculas sin incluir la primera. En caso de que las palabras sean muy extensas se utilizan abreviaturas en inglés. Se debe declarar cada variable en una línea distinta.

Ejemplos:

```
private $result;

private $objR;
```

Comentarios:

Se utilizan para especificar funciones y esclarecer alguna operación específica de algún método. La utilización de los mismos no está presente en todo el código, solo en aquellos casos que sea necesario.

Sentencias simples:

Cada línea debe contener una sola sentencia.

Ejemplo:

```
$storeResult = "cp -r tmp/".$data->name."/cflow/ ". $outputDir;

$fp = fopen( $url, "w");
```

Sentencias compuestas

Deben estar indentadas a un nivel superior que el precedente. Todas las sentencias del tipo *if, for, while, do... while* deben tener llaves. En el caso de las sentencias que posean una sola línea en su interior se pueden omitir las llaves. Debe existir un espacio entre la declaración de una sentencia y la siguiente. Las llaves deben ocupar una línea de código.

Ejemplos:

```
 if (is_bool($val))
            $val = $val? 'true' : 'false';
...
for ( $i=0; $i<count($allowedUser) ; $i++ )
   {
            if ( $user == $allowedUser[$i] )
             {
                    return true;
             }

   }
```

2.5. Desarrollo de Iteraciones

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

activamente durante esta fase del ciclo. Las iteraciones son también utilizadas para medir el progreso del proyecto.

En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor del negocio). Al final de la última iteración el sistema estará listo para entrar en producción. (Letelier 2008)

2.5.1. Implementación

Las HU agrupadas en cada iteración se van implementando durante el transcurso de la iteración a la cual pertenecen. Al principio de cada iteración se lleva a cabo una revisión del plan de iteraciones y se modifica en caso de ser necesario. Como parte de este plan se descomponen las HU en tareas de desarrollo, asignando a un grupo o una persona como responsable de su implementación. Estas tareas son para el uso estricto de los programadores por lo que pueden ser escritas en lenguaje técnico y no necesariamente entendible por el cliente.

A continuación quedan detalladas las tareas de desarrollo realizadas en cada una de las iteraciones.

Iteración 1

La primera iteración tendrá como objetivo darle cumplimiento a las HU 1,2,3,4 que representan un mayor valor para el cliente, pues con las mismas se conformará la base de la estructura del negocio. Estas recogen funcionalidades de gran importancia para el proyecto, pues a través de ellas se definen aspectos que serán utilizados luego por las demás funcionalidades.

Historia de usuario	Tiempo estimado (Semanas)	Tiempo real (Semanas)
Insertar proyecto	0.5	0.5
Consultar proyecto	0.5	0.5
Gestionar Proyecto	1	1
Gestionar Grupos de Proyectos	1	1

Tabla 24. Historias de usuarios planificadas para la primera iteración

Tareas de las historias de usuarios desarrolladas en la primera iteración

Luego de relacionar las HU pertenecientes a esta iteración, se procede a la especificación de las principales tareas de desarrollo que se realizaron para cumplir el propósito de la misma. (Anexo 1)

Iteración 2

La segunda iteración está centrada en desarrollar la HU 5, 6 que son de gran importancia también para el cliente, además de que incluyen una mayor complejidad de desarrollo, pues para realizar la integración, será necesario el trabajo con la herramienta automatizada que la propicia.

Historia de usuario	Tiempo estimado (Semanas)	Tiempo real (Semanas)
Autenticar Usuario	0.5	0.5
Realizar integración	2.5	2.5

Tabla 25. Historias de usuarios planificadas para la segunda iteración

Tareas de las historias de usuarios desarrolladas en la segunda iteración

Luego de relacionar las HU pertenecientes a esta iteración, se procede a la especificación de las principales tareas de desarrollo que se realizaron para cumplir el propósito de la misma. (Anexo 2)

Iteración 3

Esta iteración se enmarca en los restantes requerimientos, por lo que abarca las HU 7, 8, 9, 10. Se desarrollará la posibilidad de aplicar las pruebas y métricas al código, funcionalidad que posee una alta complejidad de desarrollo, pues será necesario el trabajo con las herramientas que las propician.

Historia de usuario	Tiempo estimado (Semanas)	Tiempo real (Semanas)
Aplicar métricas de calidad y pruebas a código	2	2
Consultar reporte	0.5	0.5
Descargar reporte	0.3	0.3
Consultar información de módulos	0.2	0.2

Tabla 26. Historias de usuarios planificadas para la tercera iteración

Tareas de las historias de usuarios desarrolladas en la tercera iteración

Luego de relacionar las HU pertenecientes a esta iteración, se procede a la especificación de las principales tareas de desarrollo que se realizaron para cumplir el propósito de la misma. (Anexo 3)

Las iteraciones de desarrollo sobre el sistema, permitieron que al finalizar se obtuviera un producto con todas las restricciones y características deseadas por el cliente.

Cabe destacar que el desarrollo de la solución se dividió en 3 iteraciones para ordenar la implementación, realizándose pequeñas entregas al cliente al final de cada una de ellas para recibir la aceptación del mismo, sin embargo el despliegue del producto en su entorno real de explotación no se realizará hasta terminada la última iteración. Es por esta razón que funcionalidades de gran importancia para la seguridad del software, como la autenticación de usuarios, se decidió desarrollarlas en la segunda iteración.

Consideraciones parciales

Conociendo como se lleva a cabo el flujo actual de eventos, se elaboró una propuesta del sistema. Se especificaron los usuarios que estarán relacionados con su utilización y funcionamiento. Se realizó una descripción de las HU precisando por el cliente la prioridad de cada una, definiendo así el orden en el que serán implementadas las mismas. Se cuenta con 10 HU que serán implementadas en tres iteraciones. Quedó elaborado el modelo necesario para llevar a cabo la implementación del sistema mediante la descripción de los estándares de codificación y la arquitectura utilizada. Se plantearon y describieron las tareas de desarrollo para dar cumplimiento a las funcionalidades abordadas por las HU. Se definió por medio del diagrama de despliegue la distribución física mediante la cual funcionará la aplicación y se diseñaron las pruebas de aceptación que determinan la confianza y seguridad de las funcionalidades del sistema para el cliente, la descripción de estas prueba en conjunto con los resultados se encuentran en el siguiente capítulo.

CAPÍTULO 3
APLICACIÓN DE LA SOLUCIÓN

Introducción

En el presente capítulo quedan especificados los resultados de la ejecución de las pruebas previamente diseñadas para probar las funcionalidades descritas. Además se definen las actividades necesarias para poner en funcionamiento la herramienta mediante el despliegue de la misma.

3.1. Pruebas de Aceptación

La ejecución de las pruebas previamente diseñadas, permitió evaluar las funcionalidades de la herramienta antes de pasarla a su entorno real de explotación. Los resultados de las mismas se encuentran a continuación:

Caso de prueba de Aceptación	
Número: 1	**Historia de Usuario:** 1
Nombre: Insertar proyecto	
Descripción: El caso de prueba permite al usuario desde la página “Configurar Proyectos” la posibilidad de insertar un nuevo proyecto.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Entradas/Pasos de ejecución: El usuario escoge la opción “Proyectos” del menú. -El sistema muestra la página de “Configurar Proyectos”. -El sistema permite realizar varias acciones. -El usuario selecciona la opción “Agregar Proyecto Nuevo”. -El sistema muestra la página “Agregar / Modificar Proyectos” con un formulario para introducir los datos del nuevo proyecto: • Nombre, SCM, URL, Rama, Directorio del código fuente, Lenguaje, Herramienta de construcción, Planificadores. -El usuario introduce los datos requeridos y presiona el botón “Agregar / Modificar”. -El sistema adiciona el nuevo proyecto, y muestra el mensaje “Proyecto insertado correctamente”.	
Resultado esperado: El sistema adiciona correctamente el nuevo proyecto.	
Evaluación de la prueba: Satisfactorio	

Tabla 27. Caso de prueba Insertar Proyecto

Caso de prueba de Aceptación	
Número: 2	**Historia de Usuario:** 2
Nombre: Consultar proyecto	
Descripción: El caso de prueba permite al usuario desde la página “Configurar Proyectos” la posibilidad de consultar los proyectos existentes.	
Condiciones de ejecución: El usuario debe estar autenticado. Debe existir al menos un proyecto en el sistema.	
Entradas/Pasos de ejecución: -El usuario escoge la opción “Proyectos” del menú. -El sistema selecciona de la base de datos todos los proyectos existentes. -El sistema muestra la página “Configurar Proyectos” con la lista de proyectos existentes.	
Resultado esperado: Se muestran los proyectos existentes en el sistema.	
Evaluación de la prueba: Satisfactorio	

Tabla 28. Caso de prueba Consultar Proyecto

Caso de prueba de Aceptación	
Número: 3	**Historia de Usuario:** 3
Nombre: Modificar proyecto	
Descripción: El caso de prueba permite al usuario restringido desde la página “Configurar Proyectos” la posibilidad de modificar un proyecto creado por él.	
Condiciones de ejecución: El usuario debe estar autenticado. Debe existir al menos un proyecto en el sistema. El usuario debe tener permiso para modificar el proyecto.	
Entradas/Pasos de ejecución: El usuario escoge la opción “Proyectos” del menú. -El sistema muestra la página de “Configurar Proyectos”. -El sistema permite realizar varias acciones en los proyectos insertados por el usuario. -El usuario selecciona la opción (imagen) “Editar” en uno de sus proyectos. -El sistema muestra la página “Agregar / Modificar Proyectos” con un formulario con los datos del proyecto a modificar: • Nombre, SCM, Url, Rama, Directorio del código fuente, Lenguaje, Herramienta de construcción, Planificadores. -El usuario modifica los datos y presiona el botón “Agregar / Modificar”. -El sistema actualiza los datos modificados y muestra un mensaje “Proyecto actualizado satisfactoriamente”.	
Resultado esperado: -El sistema actualiza correctamente los datos del proyecto.	
Evaluación de la prueba: Satisfactorio	

Tabla 29. Caso de prueba Modificar Proyecto

Caso de prueba de Aceptación	
Número: 4	**Historia de Usuario:** 3
Nombre: Eliminar proyecto	
Descripción: El caso de prueba permite al usuario restringido desde la página "Configurar Proyectos" la posibilidad de eliminar un proyecto insertado por él.	
Condiciones de ejecución: El usuario debe estar autenticado. Debe existir al menos un proyecto en el sistema insertado por el usuario. El usuario debe tener permiso para eliminar el proyecto.	
Entradas/Pasos de ejecución: -El usuario escoge la opción "Proyectos" del menú. -El sistema muestra la página "Configurar Proyectos". -El sistema permite realizar varias acciones en los proyectos insertados por el usuario. -El usuario selecciona la opción (imagen) "Eliminar" en uno de sus proyectos.	
Resultado esperado: El sistema elimina satisfactoriamente	
Evaluación de la prueba: Satisfactorio	

Tabla 30 Caso de prueba Eliminar Proyecto

Caso de prueba de Aceptación	
Número: 5	**Historia de Usuario:** 4
Nombre: Modificar grupos de proyectos	
Descripción: El caso de prueba permite al administrador desde la página "Configurar Proyectos" la posibilidad de modificar cualquier proyecto existente en la herramienta.	
Condiciones de ejecución: El usuario debe estar autenticado. Debe existir al menos un proyecto en el sistema.	
Entradas/Pasos de ejecución: El administrador escoge la opción "Proyectos" del menú. -El sistema muestra la página "Configurar Proyectos". -El sistema permite realizar varias acciones en todos los proyectos. -El usuario selecciona la opción (imagen) "Editar" en cualquier proyecto. -El sistema muestra la página "Agregar / Modificar Proyectos" con un formulario con los datos del proyecto a modificar: -Nombre, SCM, Url, Rama, Directorio del código fuente, Lenguaje, Herramienta de construcción, Planificadores. -El administrador modifica los datos y presiona el botón "Agregar / Modificar". -El sistema actualiza los datos modificados y muestra el mensaje "Proyecto actualizado	

satisfactoriamente”.
Resultado esperado: -El sistema actualiza correctamente los datos del proyecto.
Evaluación de la prueba: Satisfactorio

Tabla 31. Caso de prueba Modificar grupos de proyectos

Caso de prueba de Aceptación	
Número: 6	**Historia de Usuario:** 4
Nombre: Eliminar grupos de proyectos	
Descripción: El caso de prueba permite al administrador desde la página “Configurar Proyectos” eliminar cualquier proyecto existente en el sistema.	
Condiciones de ejecución: El usuario debe estar autenticado. Debe existir al menos un proyecto en el sistema.	
Entradas/Pasos de ejecución: -El administrador escoge la opción “Proyectos” del menú. -El sistema muestra la página “Configurar Proyectos”. -El sistema permite realizar varias acciones en todos los proyectos. -El administrador selecciona la opción (imagen) “Eliminar” en cualquier proyecto.	
Resultado esperado: El sistema elimina satisfactoriamente	
Evaluación de la prueba: Satisfactorio	

Tabla 32 Caso de prueba Eliminar grupos de proyectos

Caso de prueba de Aceptación	
Número: 7	**Historia de Usuario:** 5
Nombre: Autenticar Usuario	
Descripción: El caso de prueba permite al usuario autenticarse para acceder al sistema.	
Condiciones de ejecución: El usuario debe acceder a la dirección de la herramienta haciendo uso de un navegador**.**	
Entradas/Pasos de ejecución: -El usuario accede a la dirección de la aplicación a través del navegador Web. -El sistema muestra una ventana de identificación requerida para insertar el usuario y la contraseña del dominio UCI. -El usuario introduce los datos correctamente. -El sistema verifica en la base de datos y muestra la página de inicio con los permisos habilitados para el usuario.	
Resultado esperado: Se muestra la página de inicio de la herramienta.	

Evaluación de la prueba:
Satisfactorio

Tabla 33. Caso de prueba Autenticar Usuario

Caso de prueba de Aceptación	
Número: 8	**Historia de Usuario:** 6
Nombre: Realizar integración	
Descripción: El caso de prueba permite la ejecución automática de la integración.	
Condiciones de ejecución: Buildbot debe detectar la hora programada. Debe existir al menos un proyecto en el sistema para realizarle la integración.	
Entradas/Pasos de ejecución: -Buildbot testea al servidor hasta que sea la hora programada en el planificador. -Buildbot descarga los proyectos desde el SCV. -Buildbot ejecuta la integración y genera un reporte.	
Resultado esperado: -Se realiza la integración del código del proyecto y genera un reporte con los resultados.	
Evaluación de la prueba: Satisfactorio	

Tabla 34. Caso de prueba Realizar integración

Caso de prueba de Aceptación	
Número: 9	**Historia de Usuario:** 7
Nombre: Aplicar métricas de calidad y pruebas a código	
Descripción: El caso de prueba permite al sistema ejecutar los módulos de pruebas y métricas insertados en la herramienta.	
Condiciones de ejecución: Debe existir al menos un proyecto en el sistema. Debe existir al menos un módulo insertado a la herramienta.	
Entradas/Pasos de ejecución: El sistema descarga los proyectos insertados Ejecuta los módulos de pruebas por cada proyecto existente en la herramienta Genera un reporte por cada módulo en cada proyecto.	
Resultado esperado: Se ejecutan las pruebas a todos los proyectos existentes en la herramienta.	
Evaluación de la prueba: Satisfactorio	

Tabla 35. Aplicar métricas de calidad y pruebas a código

Caso de prueba de Aceptación

Número: 10	**Historia de Usuario:** 8
Nombre: Consultar reporte	
Descripción: El caso de prueba permite al usuario desde la página “Reportes” la posibilidad de consultar los reportes generados.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Entradas/Pasos de ejecución: -El usuario escoge la opción “Reportes” del menú. -El sistema muestra la página “Reportes” con un formulario para insertar los datos de los reportes a consultar. • Por nombre de Proyecto. • Por Fecha. -El usuario introduce los datos y escoge una de las dos opciones. -El sistema muestra una lista de los reportes generados según la categoría escogida. -El usuario escoge el reporte deseado. -El sistema muestra el contenido del reporte seleccionado.	
Resultado esperado: El sistema muestra el reporte correctamente.	
Evaluación de la prueba: Satisfactorio	

Tabla 36. Caso de prueba Consultar Reporte

Caso de prueba de Aceptación	
Número: 11	**Historia de Usuario:** 9
Nombre: Descargar reporte	
Descripción: El caso de prueba permite al usuario desde la página “Reportes” la posibilidad de descargar el reporte consultado.	
Condiciones de ejecución: El usuario debe estar autenticado. Se debe haber ejecutado correctamente el caso de prueba Consultar reporte.	
Entradas/Pasos de ejecución: El usuario presiona la opción descargar reporte (imagen) en la página del reporte consultado. El sistema muestra una ventana para la descarga del fichero.	
Resultado esperado: El sistema da la opción de descargar el reporte.	
Evaluación de la prueba: Satisfactorio	

Tabla 37. Caso de prueba Descargar reporte

Caso de prueba de Aceptación	
Número: 12	**Historia de Usuario:** 10
Nombre: Consultar información de módulos	
Descripción: El caso de prueba permite al usuario desde la página “Módulos” la posibilidad de consultar información sobre los módulos existentes en la herramienta.	
Condiciones de ejecución: El usuario debe estar autenticado. Debe existir al menos un módulo en el sistema.	
Entradas/Pasos de ejecución: -El usuario escoge la opción “Módulos” del menú. -El sistema muestra la página “Módulos” con la descripción de cada uno insertado en la herramienta.	
Resultado esperado: El sistema muestra la información correctamente.	
Evaluación de la prueba: Satisfactorio	

Tabla 38. Caso de prueba Consultar información de módulos

A partir de los resultados arrojados por la ejecución de las pruebas se puede llegar a la conclusión de que la herramienta se encuentra lista para ser desplegada en el CEDIN.

3.2. Despliegue de la solución

El despliegue es la realización de acciones para la ejecución de una actividad que se ha venido desarrollando con anterioridad.

El proceso de despliegue de software es la transición exitosa del sistema desarrollado, a sus usuarios. Es una etapa delicada en cualquier proyecto. Esta disciplina describe las actividades asociadas al aseguramiento para que el producto de software esté disponible para sus usuarios finales y es la culminación del esfuerzo de desarrollo de software. (Pardo y Hernández 2009)

El despliegue puede ser divido en diversas etapas que aseguran la calidad del proceso.

3.2.1. Preparación del despliegue

La preparación desempeña un papel primordial en el éxito del despliegue ya que es en este período donde se sientan las bases para el resto del proceso, garantizando de esta forma que todas las actividades implicadas en el procedimiento se realicen con la mayor organización y calidad posible. Su objetivo fundamental es preparar el escenario requerido para un despliegue exitoso.

Forman parte de este período actividades como la selección y la preparación del personal que llevará a cabo el despliegue, que para este producto quedarán omitidas debido a que el mismo será realizado enteramente por los autores de la investigación que son quienes además, poseen mayor conocimiento del funcionamiento de la herramienta así como de las condiciones necesarias para su instalación y ejecución.

En esta etapa se hizo necesaria la realización de un diagnóstico inicial en el que fueron analizadas las características de los usuarios finales del sistema, las condiciones del lugar donde será desplegada la solución y una evaluación de los recursos tecnológicos con los que cuenta el CEDIN para el despliegue.

Diagnóstico Tecnológico

Requerimientos

Para la instalación del sistema se necesita un PC conectada a la red de la universidad donde funcionarán los servidores Apache, el controlador de integración BuildMaster, los servidores de integración BuildSlaves y el servidor de base de datos.

Requerimientos mínimos de hardware:

RAM: 256 Mb
CPU: 1.0 GHz
HDD: 1Gb

Requerimientos mínimos de software:

Buildbot, Apache2, Python

Resultados

No.	Tipo de Recurso	Características	Cant.	Evaluación (Adecuado, Deficiente)
1	Computadora	Software: Buildbot Apache2	1	Adecuado

		Python Hardware: RAM:1 Gb CPU:3.0 GHz 4 HDD: 160 Gb		

Tabla 39. Evaluación de recursos tecnológicos.

Observaciones

Los requerimientos tecnológicos del servidor con que se cuenta para el despliegue satisfacen las necesidades mínimas del software, por lo que las condiciones son adecuadas para la realización del mismo.

Diagnóstico del Personal.

Requerimientos

El personal que utilizará el sistema debe tener conocimientos de informática, específicamente en cuanto al desarrollo de software se refiere, debido a que para la ejecución de las funcionalidades de la herramienta, serán necesarios parámetros que solo usuarios que conozcan del tema podrán proporcionar.

Resultados

No	Usuario	Acceso a Comp.	Conocimientos Informáticos
1	Usuarios del dominio UCI	Áreas de la Universidad.	Sí

Tabla 40. Evaluación inicial de Recursos Humanos.

Observaciones

El personal que usará la herramienta posee conocimientos necesarios para proporcionarle los parámetros para su ejecución y beneficiarse de los resultados.

Diagnóstico de las condiciones constructivas

Requerimientos

Se requiere de un lugar con buena climatización, que permita al servidor operar 24 horas toda la semana, con seguridad media para el servidor y buenas condiciones constructivas.

Resultados

No.	Elemento	Características	Estado
1	Sistemas de Seguridad	El laboratorio donde se encuentra el servidor cuenta con una puerta con llave, al que tiene acceso el personal del centro.	Adecuado
2	Sistemas de climatización	El laboratorio está debidamente climatizado.	Adecuado
3	Construcción	El laboratorio posee buenas condiciones constructivas.	Adecuado

Tabla 41. Evaluación de las condiciones constructivas.

Observaciones

Las condiciones constructivas del local con que se cuenta para el despliegue satisfacen las necesidades para su instalación, por lo que el mismo es adecuado para la realización de esta actividad.

Conclusiones del diagnóstico

Tras realizar el diagnóstico se llega a la conclusión que el Centro de Informática Industrial cuenta con las condiciones mínimas necesarias Hardware y Recursos Humanos para la realización del despliegue de la herramienta.

Otra de las actividades necesarias en esta etapa es la preparación del equipamiento, momento en el que se instala y configura el equipamiento necesario para la ejecución óptima del sistema.

Para la ejecución óptima del sistema fue necesario la instalación y configuración del Buildbot, lo que incluye el Maestro o BuildMaster, y los esclavos o BuildSlaves, fue necesaria además la instalación y configuración del servidor web Apache y el sistema gestor de base de datos MySQL.

Para la ejecución de los módulos de prueba fue necesaria también la instalación y configuración de herramientas como Cflow, Valgrind, Rats, Flawfinder, CCCC y SLOCCount.

3.2.2. Planificación del despliegue

Durante este período se ejecutan un número de acciones orientadas a la gestión del tiempo y los recursos necesarios para el desarrollo correcto del proceso de despliegue, ya que se describe y se planifica la secuencia y duración de las actividades para asegurar que se ejecuten correctamente, en el tiempo previsto, y con los recursos estimados, permitiendo que se mitiguen riesgos.

Para el despliegue de esta herramienta se definió un cronograma que permitirá el desarrollo de las actividades pertenecientes a cada etapa. (Anexo 5)

3.2.3. Implantación Piloto

El Piloto constituye el primer acercamiento real de los usuarios finales al sistema. Durante esta etapa, surgen un número considerable de no conformidades que son resueltas por el equipo de desarrollo en paralelo al resto de las actividades. Se hace énfasis en la etapa de pruebas, con un alto nivel de importancia. El desarrollo de la solución va a iterar con el proceso de pruebas debido a que se deben solucionar los errores, cambios y no conformidades, y posteriormente una nueva versión y revisión del producto final.

En esta etapa se decidió instalar y ejecutar el software en la entidad cliente durante un período no mayor a una semana, en la que se le realizaron diversas pruebas que permitieron a los usuarios finales una mayor aceptación de la herramienta.

Para la puesta en marcha del piloto fue instalado y configurado cada software necesario para la utilización del sistema. Luego de comprobar el correcto funcionamiento de los mismos se procedió a la instalación piloto de Sherlock.

La ejecución nuevamente de las pruebas de aceptación diseñadas con anterioridad, permitió conocer que la herramienta funciona correctamente en un entorno de explotación, además de que los clientes quedaron satisfechos con los resultados.

Antes de pasar a liberar la versión final del software, se revisó la documentación y quedó aprobado formalmente el producto.

3.2.4. Ejecución del despliegue

Para la puesta en marcha de la herramienta fue instalado y configurado cada software mencionado anteriormente, cada uno necesario para la utilización de la herramienta. Luego de comprobar el correcto funcionamiento de los mismos se procede a su instalación.

Luego de instalada la herramienta, fue realizado un monitoreo para controlar el sistema y verificar que el mismo funciona correctamente, a su vez, se le realizaron pruebas de aceptación en el entorno de producción, las que permitieron probar el correcto funcionamiento de la herramienta en el entorno real de explotación, en dichas pruebas se ejecutaron nuevamente los casos de pruebas diseñados con anterioridad obteniendo la conformidad del cliente con la versión estable del producto.

Se consideró innecesaria realizar una capacitación a los usuarios finales, debido a que los mismos poseen conocimiento de informática y en provecho de esta característica, se creó un manual de usuario que explica cómo usar la herramienta.

3.2.5. Finalización del despliegue

En esta etapa, se irá incrementando gradualmente el número de personas que comiencen a trabajar y a utilizar el sistema de la forma más real posible, estarán conectados la mayor cantidad de usuarios de diversos roles, y responsabilidades. Se debe verificar si no existe nada pendiente, logrando la meta deseada, y la satisfacción total de los clientes.

Para la finalización de la etapa de despliegue se definió un período de 3 semanas de acompañamiento a los usuarios en el que se aclararán las dudas que los mismos puedan presentar en el desarrollo de las actividades normales con el software.

3.3. Resultados de la herramienta sobre los proyectos del CEDIN

Finalizado el despliegue, diversos proyectos del centro se vieron beneficiados con el uso de la herramienta, fueron insertados a la misma distintos módulos de la línea de procesamiento en tiempo real del proyecto middleware del SCADA.

Entre los módulos insertados se encuentran: **libicedatatypes-2.0.1**, **libasynccomm-2.0.1, libsynccomm-2.0.1, libasynccomm-1.0.0, libicedatatypes-1.0.0, libsynccomm-1.0.0.**

Con la ejecución de la herramienta sobre estos módulos, fue posible la integración del código perteneciente a cada uno de ellos, generándose paquetes de Debian en libicedatatypes y libasynccomm así como reportes de la ejecución de los módulos de pruebas que aportaron datos relevantes para el trabajo con dicho código.

Consideraciones parciales

Se ejecutaron las pruebas de aceptación que permitieron demostrar que el sistema se encuentra listo para ser desplegado en el CEDIN. Además se realizaron las actividades pertinentes para desplegar la solución en el centro, dividiéndose el proceso en las 5 etapas mencionadas en el desarrollo del capítulo.

CONCLUSIONES

Una vez finalizada la investigación se puede concluir que:

- Se realizó un análisis de las herramientas y tecnologías utilizadas para llevar a cabo el proceso de integración, prueba, y verificación de métricas al software.
- Se logró la generación de los artefactos propuestos por la metodología seleccionada para el desarrollo de la solución.
- Quedó implementada la herramienta informática propuesta.
- Fue probado el sistema, lo que permitió demostrar el cumplimiento de las exigencias del cliente.
- La herramienta quedó desplegada en el CEDIN para su explotación por parte de los usuarios finales.
- Se elaboró un manual que muestra el uso de la herramienta. La disponibilidad de este manual supone la completitud de los conocimientos necesarios para el trabajo con la misma.
- La herramienta permite satisfactoriamente la integración continua, prueba y la verificación de métricas de calidad al código fuente de los proyectos del CEDIN.

RECOMENDACIONES

Concluido el desarrollo de este trabajo se recomienda:

- Incluir un panel de administración que permita la gestión de la información de los usuarios que utilizan la herramienta.
- Incluir nuevos módulos que ejecuten pruebas y apliquen métricas al código para aprovechar la extensibilidad del sistema.
- Mejorar el sistema aplicando nuevas tecnologías y explotarlas al máximo para evitar que la tecnología con la cual fue desarrollado se vuelva obsoleta, logrando un mejor funcionamiento.
- La utilización de esta experiencia como material de estudio para el desarrollo de aplicaciones similares por parte de otros desarrolladores.

REFERENCIAS BIBLIOGRÁFICAS

Gómez, Wilson Javier Argüello. MDS 360° Metodología de desarrollo de software un enfoque práctico y global versión 1.0.11 BETA (20071020). 2007. [Citado 19 Mayo 2011]. Disponible en: http://www.otcolombia.com/documentos/mds360-1.0.11-beta.pdf.

Microsoft. Build and Deploy Continuously. 2010. [Citado 28 Abril 2011]. Disponible en: http://msdn.microsoft.com/en-us/library/ee308011.aspx.

Fowler, Martin. Continuous Integration. Mayo 2006. [Citado 5 Enero 2011]. Disponible en: http://martinfowler.com/articles/continuousIntegration.html.

Fernández, José Antonio Calvo. Entorno para la ejecución de pruebas de integración de la plataforma eBox. Noviembre 2007. [Citado 15 Diciembre 2010]. Disponible en: http://people.warp.es/~josh/memoria.pdf.

Apache Continuum. *Sitio Oficial de Apache Continuum* Febrero 2011. [Citado 23 Mayo 2011]. Disponible en: http://continuum.apache.org/.

Hudson. *Sitio Oficial de Hudson*. Mayo 2011. [Citado 23 Mayo 2011].]. Disponible en: http://hudson-ci.org.

Buildbot. Sitio Oficial de Buildbot. 2011. [Citado 24 Febrero 2011]. Disponible en: http://trac.buildbot.net/wiki/AboutBuildbot.

Sordo, Mohamed. Sistema de compilación y testeo automatizado, distribuido y multiplataforma. Junio 2006. [Citado 16 Enero 2011]. Disponible en: http://www.dtic.upf.edu/~parumi/tmp/mohamed-testfarm-memoria.pdf.

Pressman, Roger S. *Ingeniería de Software un enfoque práctico*. 5ta Edición 1998.

Galiano, Fernando Berzal. *Diseño de arquitecturas software*. 2005.

Prado, Elena Raja. *Actas de Talleres de Ingeniería del Software y Bases de Datos*. 2007. Vol. 1.

REFERENCIAS BIBLIOGRÁFICAS

García, Félix. Proceso Software y Gestión del Conocimiento. 2008. [Citado 18 Marzo 2011]. Disponible en: http://alarcos.inf-cr.uclm.es/doc/psgc/doc/psgc-4a.pdf.

Doria, Heidi González. Las Métricas de Software y su Uso en la Región. 2001. [Citado 2 Febrero 2011]. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/portada.html.

Mendoza, Gonzalo Mena. ISO 9126-3: Métricas Internas de la Calidad del Producto de Software. Marzo 2006. [Citado 24 Mayo 2011]. Disponible en: http://mena.com.mx/gonzalo/maestria/calidad/presenta/iso_9126-3/.

Letelier, Patricio. Metodologías Ágiles en el Desarrollo de Software. 2008. [Citado 20 Enero 2011]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.4553&rep=rep1&type=pdf#page=9.

Cáceres, Paloma, y Esperanza Marcos. Procesos Ágiles para el desarrollo de Aplicaciones Web. Octubre 2001. [Citado 5 Febrero 2011]. Disponible en: http://altea.dlsi.ua.es/webe01/articulos/s112.pdf.

KRUCHTEN, P. The Rational Unified Process: An Introduction. 2000. [Citado 5 Marzo 2011]. Disponible en: http://books.google.com/books?id=RYCMx6o47pMC&dq=The+Rational+Unified+Process:+An+Introduction+2002&printsec=frontcover&source=bn&hl=es&ei=4sfIS7rFFoaglAfU4YTqCQ&sa=X&oi=book_result&ct=result&resnum=4&ved=0CB8Q6AEwAw#v=onepage&q=The%20Rational%20Unified%20Process%3A%20An%20Introduction%202002&f=false.

Molpeceres, Alberto. Procesos de desarrollo: RUP, XP y FDD. Diciembre 2002. [Citado 5 Febrero 2011]. Disponible en: http://www.willydev.net/descargas/Articulos/General/cualxpfddrup.PDF.

Beck, Kent. Extreme Programming Explained. Septiembre 1999. [Citado 7 Febrero 2011]. Disponible en: http://www.mip.sdu.dk/~brianj/Extreme%20Programming%20Explained%20-%20Kent%20Beck;%20Addison-Wesley,%201999.pdf.

Hernán, Schenone Marcelo. Diseño de una Metodología Ágil de Desarrollo de Software. 2004. [Citado 16 Enero 2011]. Available from world wide web: http://materias.fi.uba.ar/7500/schenone-tesisdegradoingenieriainformatica.pdf.

REFERENCIAS BIBLIOGRÁFICAS

Lafuente, Guillermo Javier. UML Unified Modeling Languaje. Febrero 2002. [Citado 24 Mayo 2011]. Disponible en: http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html.

Hinostroza, Raúl Rodas. LinuxCentro.net - Características de PHP. 2005. [Citado 14 Febrero 2011]. Disponible en: http://www.linuxcentro.net/linux/staticpages/index.php?page=CaracteristicasPHP.

Álvarez, Miguel Ángel. Manual de CSS, hojas de estilo. 2000. [Citado 14 Febrero 2011]. Disponible en: http://www.desarrolloweb.com/manuales/manual-css-hojas-de-estilo.html.

Bluefish Editor. Sitio Oficial de Bluefish Editor. 2011. [Citado 20 Febrero 2011]. Disponible en: http://bluefish.openoffice.nl/index.html.

Pérez, Iván Nieto. Tutorial de JavaScript: introducción al lenguaje. 2006. [Citado 3 Abril 2011]. Disponible en: http://www.elcodigo.net/tutoriales/javascript/javascript1.html.

Hernandis, J. A. Why Visual Paradigm for UML? 2005. [Citado 9 Enero 2011]. Disponible en: http://www.visual-paradigm.com/product/vpuml/>.

Castellanos, Ma. Argenis González, y Wilson Rojas Pabón. Comparación entre sistemas de gestión de bases de datos (SGBD) bajo licenciamiento libre y comercial. 2005. [Citado 14 Febrero 2011]. Disponible en: http://www.ilustrados.com/documentos/sgbd.pdf.

Sussman, Ben Collins, Brian W. Fitzpatrick, y C. Michael Pilato. Control de versiones con Subversion. 2002. [Citado 27 Marzo 2011]. Disponible en: http://svnbook.spears.at/nightly/es/svn-book.html.

Gayo, José Emilio Labra, Daniel Fernández Lanvin, Jesús Calvo Salvador, y Agustín Cernuda del Río. Una Experiencia de aprendizaje basado en proyectos utilizando herramientas colaborativas de desarrollo de software libre. 2006. [Citado 2 Febrero 2011]. Disponible en: http://www.di.uniovi.es/~labra/FTP/Papers/LabraJenui06.pdf.

Iglesias, Francisco Javier Aguirre. Halotis IDE. Septiembre 2009. [cited 10 Febrero 2011]. Available from world wide web: http://javaguirre.net/wp-content/uploads/code/qt-halotis.odt.

REFERENCIAS BIBLIOGRÁFICAS

Joskowicz, José. Reglas y Prácticas en eXtreme Programming. 2008. [Citado 5 Febrero 2011]. Disponible en: http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

Escribano, Gerardo Fernández. Introducción a Extreme Programming. 2002. [Citado 5 Febrero 2011]. Disponible en: http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-XP.pdf.

Casas, Sandra, y Héctor Reinaga. Identificación y Modelado de Aspectos Tempranos dirigido por Tarjetas de Responsabilidades y Colaboraciones. 2008. [Citado 20 Febrero 2011]. Disponible en: http://www.oocities.org/espanol/profeprog2/INVPAPER25.pdf.

Jacobson, Ivar, Grady Booch, y James Rumbaugh. *El Proceso Unificado de Desarrollo de Software*. 2000 [Citado 17 Marzo 2011]. Disponible en: http://bibliodoc.uci.cu/pdf/8478290362.pdf.

Torre, César de la, Unai Zorrilla, Ramos, Miguel Ángel y Calvarro, Javier. *Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0*. 1ra edición Marzo 2010.

Rojas, Mc Juan Carlos Olivares. Patrones de Diseño. Mayo 2007. [Citado 25 Abril 2011]. Disponible en: http://antares.itmorelia.edu.mx/~jcolivar/courses/dp07b/patrones.pdf.

Fernández, Carlos. Estándar para codificación en lenguaje C++. 2008. [Citado 25 Abril 2011]. Disponible en: http://progra.iteso.mx/estandares/estandar%20codificacion%20c%2B%2B/estandarcodificacion.pdf.

Pardo, Daily Miranda, y Juniel Tamayo Hernández. *Procedimiento para el despliegue de soluciones de software desarrolladas en la Universidad de las Ciencias Informáticas, basado en casos de estudio*. Ciudad de la Habana. 2009.

GLOSARIO

Software: Programas, procedimientos y reglas para la ejecución de tareas específicas en un sistema de cómputo.

Calidad de Software: conjunto de propiedades inherentes a un objeto que le confieren capacidad para satisfacer necesidades implícitas o explícitas.

Caso de Prueba: Conjunto de condiciones o variables bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio.

PDF: Formato de documento portátil (del inglés *portable document format*)

No Conformidades: Un no cumplimiento a un requisito.

Protocolo: Conjunto de reglas usadas por computadoras para comunicarse unas con otras a través de una red.

HTTP: Protocolo para la conexión segura.

IRC: (Acrónimo del inglés *Internet Relay Chat*) es un protocolo de comunicación en tiempo real basado en texto, que permite debates entre dos o más personas.

TCP: Protocolo de Control de Transmisión (del inglés *Transmission Control Protocol)*

Capas: Se ocupan de la división lógica de componentes y funcionalidad y no tienen en cuenta la localización física de componentes en diferentes servidores o en diferentes lugares.

CPU: Unidad de procesamiento central (del inglés Central Process Unit)

SCADA: Control Supervisor y Adquisición de Datos (del inglés Supervisory Control And Data Acquisition).

COCOMO: Modelo Constructivo de Costes (del inglés COnstructive COst MOdel) es un modelo matemático de base empírica utilizado para estimación de costes de software.