Universidad de las Ciencias Informáticas

Facultad 3

Sistema informático para el control de cambios durante el proceso de desarrollo de software en los proyectos de CEGEL

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor: Juan Carlos López Acosta

Tutores: Ing. Denia Madruga Hernández

Ing. Juniel Tamayo Hernández

La Habana, mayo de 2016.

“Año del 58 aniversario del triunfo de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Centro de gobierno electrónico (CEGEL) 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 _____.

_____ _____

Juan Carlos López Acosta

Autor

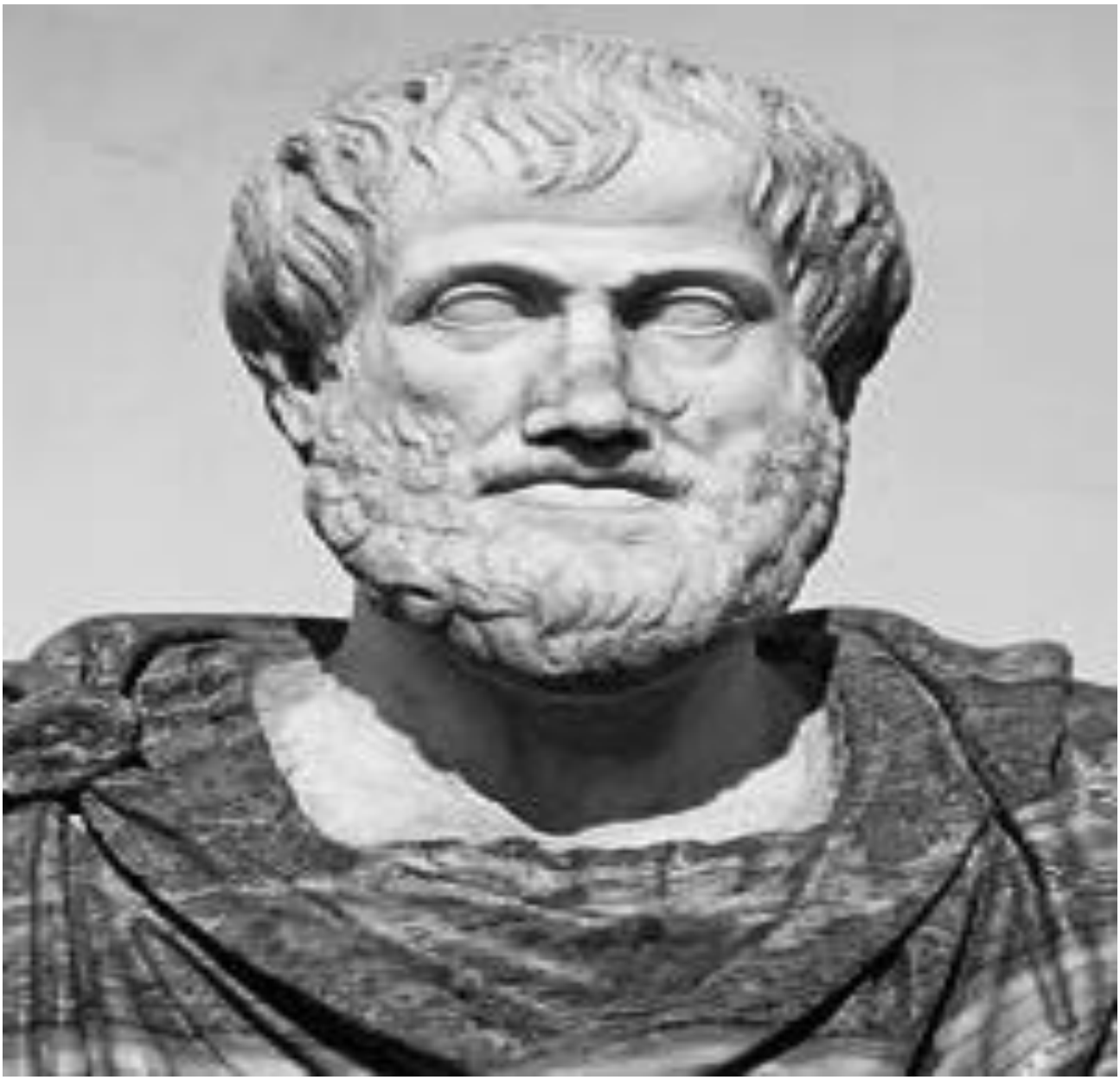
Juniel Tamayo Hernández

Tutor

_____

Denia Madruga Hernández

Tutor



Las raíces de la educación son amargas, pero la fruta es dulce.

Aristóteles

Agradecimientos

A mis padres Dalgis y Juan Carlos por darme la vida y por educarme.

A mi familia por todo el apoyo que siempre me han brindado.

A mis tutores Juniel y Denia por su ayuda incondicional.

A mis hermanos y amigos que a lo largo de estos años han estado siempre a mi lado, en las malas y en las buenas.

A todos los profesores de la facultad 3.

A todos los héroes y mártires que luchan y lucharon por esta magnífica obra que es la Revolución.

A nuestro Comandante en Jefe Fidel Castro Ruz por ser el creador de este sueño.

Dedicatoria

Le dedico este trabajo a mi tío Alfredo, que aunque ya no se encuentra con nosotros siempre estará presente en mi corazón.

Resumen

Cuba apuesta por el desarrollo de software como una vía de ingresos a la economía nacional, para esto se crean equipos de trabajo compuesto por diferentes miembros que juegan diversos roles en la organización. La sincronización del trabajo de cada uno de ellos no es una tarea fácil, lo que crea una oportunidad para los sistemas de apoyo a la gestión de la configuración. El control de cambios es una rama fundamental dentro de la gestión de la configuración y uno de los procesos más complejos a lo largo de todo el proceso de desarrollo de software.

La presente investigación surgió por la necesidad de contar en los proyectos productivos del centro de CEGEL en la Facultad 3, con una herramienta que permitiera al equipo de desarrollo llevar el control de los cambios durante todo el proceso de desarrollo del software.

Para ello se crea un sistema empleando la metodología AUP, siguiendo el subproceso de control de cambios perteneciente al programa de mejoras, y con el empleo en todo el proceso de desarrollo de herramientas y bibliotecas pertenecientes al movimiento del software libre.

PALABRAS CLAVE

Calidad del proceso de software, Gestión de la configuración, PostgreSQL, Sistema de Control de Cambios.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Introducción

El proceso de desarrollo de software es una tarea compleja, que vincula un equipo de trabajo integrado por miembros, los cuales, juegan diferentes roles dentro de dicho proceso. A consecuencia, las actividades realizadas durante el desarrollo de software son por lo general muy dinámicas y los productos generados tienen una alta susceptibilidad al cambio, por lo que es necesario llevar a cabo un adecuado control sobre cada uno de los elementos (códigos fuente, ejecutables, documentos y datos) y proceso con que se trabaja. Los cambios se pueden presentar por diferentes motivos, variación de los requisitos, o por errores cometidos durante el proceso de desarrollo; esto denota la necesidad de establecer un seguimiento sobre los productos de trabajo.

Gestión de Cambio (MOC, por sus siglas en inglés) es una técnica de uso muy común. (MANUELE 2012) describe que los objetivos de la gestión del cambio son:

• Identificar las posibles consecuencias de un cambio de procesos.

• Planificar con antelación, de modo que se puedan adoptar medidas adecuadas, antes de que se produzca un cambio, y continuamente cuando el cambio avance.

• Se identifiquen y analicen los peligros, y se evalúen los riesgos.

• A fin de manejar niveles aceptables de riesgos, se tomen decisiones adecuadas para evitarlos, eliminarlos o controlarlos, y que se los mantenga durante el proceso de cambio.

Uno de los principales problemas que está enfrentando la industria de software es la ineficiencia con que se gestionan los cambios, a pesar de existir procesos definidos para ello su uso se hace bastante complejo. La gestión de los cambios en algunos casos se convierte en un peso adicional para el equipo de desarrollo y una mala realización de esta actividad pudiera afectar el producto final, ya sea en su calidad o en su fecha de terminación.

La Universidad de Ciencias Informáticas (UCI), actualmente participa en la creación de muchos productos de software para la informatización de la sociedad cubana y la exportación. La política institucional de la universidad hace uso del Modelo CMMI-DEV v1.3, el cual constituye una guía para aplicar las mejores

prácticas en una entidad desarrolladora y de esta forma garantizar la calidad de sus productos y procesos. CMMI cuenta con diferentes áreas, entre ellas, la de gestión de la configuración, donde se encuentra el subproceso de control de cambios.

Hoy en el centro de Centro de Gobierno Electrónico (CEGEL) de la facultad 3, perteneciente a la universidad, al no existir como guía un sistema automatizado que cumpla con el proceso que rige CMMI en su programa de mejora para llevar a cabo el control de los cambios, se obtiene como resultado que los proyectos estén expuestos a riesgos relacionados con esta disciplina. He aquí la importancia de implantar un sistema de control de cambios, que ejecute el proceso institucionalizado y cumpla con las políticas que el mismo propone. Estas políticas son:

Dar seguimiento a las solicitudes de cambios:

- Registrar y analizar las solicitudes de cambio para determinar el impacto del cambio en el producto de trabajo.
- Las solicitudes de cambio deben monitorearse hasta su cierre.

En estas políticas se hace mención de una actividad muy importante, el análisis de impacto de una solicitud. En CEGEL, al realizarse una petición de cambio, se debe consultar una documentación amplia para poder comprender la repercusión que conlleva realizar el cambio, lo que hace de este análisis un proceso embarazoso que puede tributar a cometer errores durante su ejecución.

Otro punto importante es el envío de notificaciones de cada solicitud. Esta actividad en el centro hoy depende del factor humano, lo que provoca que en muchas ocasiones se olvide informar a algún interesado, la notificación se realice pero no en el momento requerido, o simplemente no se notifique. Esto puede provocar pérdida de tiempo y esfuerzo por parte del equipo de trabajo al encontrarse trabajando sobre elementos ya no existentes o modificados.

Se puede mencionar además que un sistema de control de cambios facilitaría la toma de decisiones del proyecto y la gestión en general del proceso, pues para tomar decisiones no necesariamente se requiere de todo el personal reunido en un mismo local.

De la situación problemática anteriormente planteada se evidencia el siguiente **problema a resolver**: ¿Cómo realizar el seguimiento durante el proceso de desarrollo de software y tribute a elevar la calidad del proceso de desarrollo de software?

El **objeto de estudio** es la gestión de la configuración en el proceso de desarrollo de software.

El **campo de acción** donde se enmarca la investigación es el control de cambios durante el proceso de desarrollo de software.

Para resolver el problema identificado se propone el siguiente **objetivo general**: desarrollar un sistema informático para el control de cambios durante el proceso de desarrollo de software en los proyectos de CEGEL que tribute a elevar la calidad del proceso de desarrollo de software.

Los **objetivos específicos** para conseguir el objetivo general son:

1. Elaborar el marco teórico de la investigación sobre el control de cambios durante el proceso de desarrollo de software.
2. Aplicar un diagnóstico del proceso de desarrollo de software en los proyectos de CEGEL.
3. Desarrollar un sistema para el control de los cambios durante el proceso de desarrollo de software en los proyectos de CEGEL, teniendo en cuenta los procedimientos definidos en el centro para esta actividad.
4. Verificar el sistema desarrollado a partir de la ejecución de pruebas de software.

Para dar cumplimiento a los objetivos específicos anteriormente planteados se definen las siguientes Tareas de la investigación:

- Caracterización del estado del arte del control de cambios durante el proceso de desarrollo de software, métodos y herramientas.
- Caracterización de las herramientas y tecnologías a utilizar para el desarrollo del producto.
- Obtención de los requisitos del sistema.

- Realización del diseño del sistema.
    - Implementación del sistema.
    - Diseño de Casos de Prueba.
    - Validación del sistema a través de la ejecución de pruebas de caja negra.

Los métodos de investigación empleados son:

Métodos teóricos:

Análisis-síntesis: se realizó un estudio de la bibliografía consultada para profundizar y realizar valoraciones sobre los fundamentos teóricos de la evolución y actualidad del control de cambios.

Histórico-Lógico: para desarrollar un estudio del estado del arte de la problemática planteada y revisar las diferentes herramientas y tendencias para el control de cambios.

Modelación: este método fue empleado para realizar los diagramas necesarios en el desarrollo y la mejor comprensión de la aplicación.

Métodos Empíricos:

Entrevista: con el propósito de conocer cómo se lleva a cabo el control de cambios en los proyectos de CEGEL y de esta forma saber las deficiencias que presentan respecto al tema.

Medición: este método se tiene en cuenta con las pruebas que se le realizan al software y las métricas.

El documento estará estructurado de la manera siguiente:

Capítulo 1 Fundamentación Teórica: el capítulo realiza un estudio preliminar de los sistemas utilizados para el control de cambio durante el proceso de desarrollo del software y aborda acerca de las tecnologías y herramientas para el desarrollo de la propuesta.

Capítulo 2 Arquitectura, diseño e implementación del sistema: el capítulo realiza una descripción del negocio a informatizar e información manejada y describe los requisitos y casos de uso del sistema. Se realiza un análisis de la solución propuesta para dar respuesta a la problemática planteada.

Capítulo 3 Verificación y Validación: en este capítulo se verificarán las variables planteadas en el sistema y se pondrán en práctica las diferentes métricas que define las pruebas para validar la propuesta de solución.

Capítulo 1. Fundamentación teórica

1.1 INTRODUCCIÓN

El presente capítulo tiene como objetivo introducir los principales conceptos y términos utilizados en la Gestión de la Configuración del Software. Será abordado con mayor profundidad, el proceso de Gestión de Cambios, tema central de la investigación, se brindan detalles de las herramientas que automatizan este proceso, para finalmente hacer una descripción de la producción en la Facultad 3. Además, se describen las herramientas y tecnologías que se emplearon para el desarrollo del sistema.

1.2 GESTIÓN DE LA CONFIGURACIÓN. CONCEPTOS

A continuación se enuncian diferentes definiciones brindadas por los autores sobre Gestión de Configuración de Software (GCS).

Babich plantea que “Al arte de coordinar el desarrollo de software para minimizar la confusión se denomina Gestión de Configuración. La Gestión de Configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. El objetivo es maximizar la productividad minimizando los errores” (BABICH 1986).

Otra definición es la que propone la Norma ISO 9000-3:1991, donde se establece que la Gestión de Configuración de Software provee mecanismo para identificar, controlar y dar seguimiento a cada una de las versiones de los elementos que conforman al producto software (BAMFORD 1995).

En el libro “*Software Reuse: Architecture, Process, and Organizartions for Busisness Success*” se plantea que la Gestión de Configuración es: Proceso de soporte cuyo propósito es identificar, definir y almacenar en una línea base los elementos de software; controla los cambios, reporta y registra el estado de los elementos y de las solicitudes de cambio; asegura la completitud, consistencia y corrección de los elementos; controla, almacena, maneja y libera los elementos asociados al producto de software” (JACOBSON 2000).

Para Rational la GCS “describe la estructura del producto e identifica los elementos que lo constituyen y que son tratados como entidades que pueden ser puestas bajo control de versiones en el proceso de GCS. La GCS tiene que ver con la definición de la configuración así como la construcción, el etiquetado y recolección de versiones de los artefactos” (RATIONAL 2003).

Refiriéndose al tema Pressman plantea que la Gestión de configuración es el arte de identificar, organizar, y controlar las modificaciones que sufre el software que construye un equipo de programación. La meta es maximizar la productividad minimizando los errores (PRESSMAN 2005).

La definición que brinda IEEE sobre GCS plantea que “Gestión de Configuración es la disciplina que abarca todo el ciclo de vida de la producción de software y productos asociados. Específicamente, requiere de la identificación de los componentes a controlar y la estructura del producto, controla todos los cambios sobre los elementos y garantiza mecanismos para auditar todas las acciones” (IEEE. Glosario de Terminología Informática 2009).

Una vez revisadas en la bibliografía estas definiciones, se puede resumir que es una disciplina aplicada a lo largo de todo el proceso de desarrollo de software, cuya misión es llevar a cabo el control de los cambios que puedan sufrir los elementos de configuración que va produciendo el proceso de ingeniería de software.

1.3 SISTEMAS DE CONTROL DE CAMBIO

En la actualidad, las empresas desarrolladoras de software cuentan con equipos de trabajo para lograr la creación de determinado producto; por lo que es necesario llevar el control y el registro de cada uno de los cambios realizados; y de esta forma:

- Reducir errores producidos en el proceso de desarrollo.
- Aumentar la calidad y la productividad.
- Acarrear una incorrecta sincronización de dichos cambios, al afectar otros elementos del sistema o las tareas realizadas por otro miembro del equipo.

He aquí la necesidad de uso de un sistema de control de cambio, capaz de llevar el seguimiento de las actualizaciones hechas por cada miembro del grupo de trabajo, sobre los muchos elementos de configuración que conforman una aplicación informática.

A continuación, la definición de control de cambios de varios autores conocedores del tema:

El sistema de control de cambios es un conjunto de actividades diseñadas para gestionar el cambio al identificar los productos de trabajo que probablemente cambien, establecer relaciones entre ellos, definir mecanismos para gestionar diferentes versiones de estos productos de trabajo, controlar los cambios impuestos y auditar e informar los cambios realizados (PRESSMAN 2005).

Un sistema de control de cambios es una herramienta de software que administra el acceso a un conjunto de archivos que componen un proyecto y su principal propósito es registrar información del usuario que realizó algún cambio (DONATO 2009; GONZÁLEZ 2008).

En la UCI, se tiene definido como sistema de control de cambios al conjunto de actividades que controlan las solicitudes de cambios emitidas a los proyectos con el objetivo de realizar la solicitud de cambio de mejora a través del producto de trabajo solicitud de cambio de mejora que es almacenado en el expediente de proyecto (MEJORAS 2015).

El presente trabajo tomará como referencia el concepto de (MEJORAS 2015) por ser el que mejor se ajusta a la investigación.

La Gestión de Cambios constituye la actividad de Gestión de Configuración más importante y su objetivo es “proporcionar un mecanismo riguroso para controlar los cambios” (ANTONIO 2001). Es decir que es preciso estar preparado para enfrentar el cambio de manera adecuada, en el momento que sea preciso, pues es inevitable. Aunque muchas veces las solicitudes de cambio se producen durante la fase de mantenimiento del producto, esta puede aparecer en cualquier momento durante el ciclo de vida del software.

(ANTONIO 2001) considera dos tipos fundamentales de cambios:

• Corrección de un defecto: es la forma en que es visto generalmente por los clientes.

• Mejora del sistema: así lo ven los desarrolladores.

Sobre esto es preciso señalar otra causa que pudiera generar la necesidad de un cambio: un cambio en el negocio que provoque la necesidad de cambiar los requisitos del sistema en cuestión. Un cambio se puede realizar a diferentes niveles,(ANTONIO 2001) menciona los siguientes:

• Informal: antes de que un Elemento de Configuración del Software pase a formar parte de una línea base, aquel que haya desarrollado el Elemento de Configuración del Software podrá realizar cualquier cambio justificado por el proyecto (siempre que no impacte en otros requisitos del sistema más amplios) sobre él.

• Al nivel del proyecto o semi-formal: una vez que un Elemento de Configuración del Software pasa la revisión técnica formal y ha sido aprobado y por tanto se ha convertido en una línea base, para que el encargado del desarrollo pueda realizar un cambio debe recibir la aprobación de:

1. El jefe del proyecto, si es un cambio local.
2. El Comité de Control de Cambios (o Autoridad de Control de Cambios), si el cambio tiene algún impacto sobre otros Elementos de Configuración del Software.

• Formal: Se adopta cuando se ha distribuido el producto al cliente, o sea cuando se empieza a comercializar. Todo cambio deberá ser aprobado por el Comité de Control de Cambios.

El nivel de burocracia del proceso de Gestión de Cambios, puede causar incomodidad a los que no estén acostumbrados, más aún cuando se describa en detalle la forma de llevarlo a un nivel formal. Pero para (PRESSMAN 2005), esto debe ser normal, ya que “sin la protección adecuada, la Gestión de Cambios puede demorar el progreso y crear un papeleo innecesario”. Es necesario establecer de forma precisa, al comienzo de cada proyecto, cuál será el proceso de Gestión de Cambios que se va a utilizar, para ello se deben definir:

• Políticas a nivel organizativo que promuevan las actividades de Gestión de Cambios.

• Los estándares que se van a adoptar y a los que será necesario ajustarse.

• Los procedimientos que se van a utilizar para poner en práctica las políticas (ANTONIO 2001).

Sobre el análisis de este último punto se hablara en el siguiente epígrafe, pues es muy importante definir un proceso a seguir una vez dadas las condiciones que generan la necesidad de hacer una solicitud de cambio.

1.4 PROCESO DE GESTIÓN DE CAMBIOS

El proceso de Gestión de Cambios en un determinado proyecto, define los pasos a seguir una vez que se dan algunas de las condiciones que generan la necesidad de hacer una solicitud de cambio. Hasta el momento se han definido varios procedimientos para la Gestión de los Cambios, (PRESSMAN 2005) describe de la siguiente manera los pasos a seguir para llevar a cabo la Gestión de Cambios Formal:

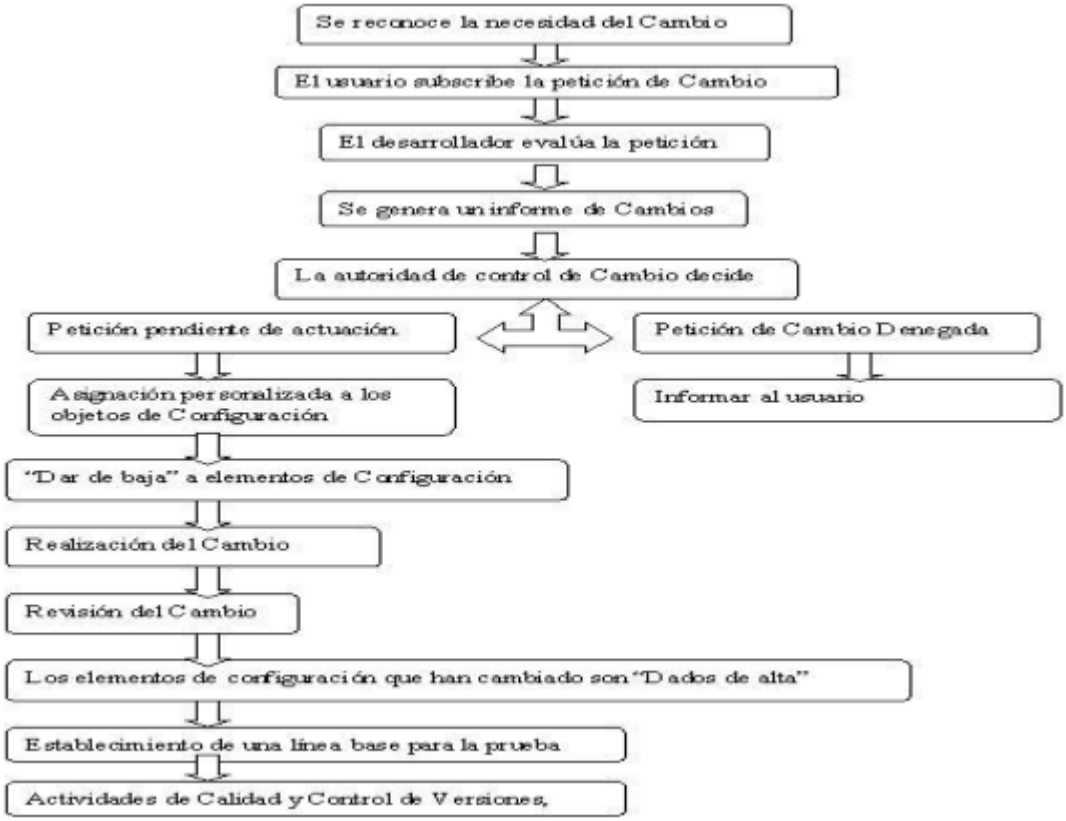


Figura 1: Proceso de gestión de cambio (PRESSMAN 2005).

Otro procedimiento existente es el que propone Angélica de Antonio, en este se explica cómo se debe llevar a cabo la Gestión de Cambios formal sobre una línea base existente. Este procedimiento incluye un paso

final ausente en el descrito por (PRESSMAN 2005): la notificación al originador del cambio luego de la realización del mismo.

Figura 2: Proceso de Gestión de Cambios [Antonio 2001].

En la actualidad, la UCI hace uso de las buenas prácticas que establece CMMI y utiliza para la parte de la gestión de procesos, las áreas que define CMMI-DEV v1.3 para el nivel 2.

Cada área tiene un grupo de subprocesos y actividades asociadas. Particularmente, en el área de (CM de sus siglas en inglés, Gestión de la configuración), se encuentra el subproceso Control de Cambios, que es donde se controlan las solicitudes de cambios emitidas a los proyectos y tiene como objetivo realizar la solicitud de cambio de mejora a través del producto de trabajo que es almacenado en el expediente de proyecto. En la Figura 3: Diagrama del Control de Cambios de CMMI, se muestra un diagrama con el flujo de las actividades que se llevan a cabo en dicho subproceso.

Comienza con un documento de solicitud de cambio de mejora, el cual pudo haber sido emitido anteriormente. En este caso el paso a seguir sería agregar a dicha solicitud el nuevo interesado. Si no fue emitida entonces el administrador de configuración registra la solicitud de cambio en la herramienta de gestión de proyectos. Posteriormente el Comité de Control de Cambios se reúne y realiza un análisis del

impacto de la solicitud de cambio de mejora emitida, teniendo en cuenta un grupo de elementos que apoyan la decisión con respecto al cambio. Luego toma la decisión de aprobar o rechazar el cambio y comunica a los interesados. Al concluir la reunión del Comité de Control de Cambios se emite una minuta de reunión donde se recogen las actividades principales de la reunión y la decisión tomada acerca de la solicitud de cambio de mejora, si la solicitud de cambio de mejora fue aprobada, el Comité de Control de Cambios asigna la implementación del cambio al equipo de proyecto involucrado. Luego el equipo de proyecto realiza la implementación del cambio, según la asignación realizada por el Comité de Control de Cambios. El administrador de la calidad revisa la implementación del cambio, en caso de no haber no conformidades se procede a actualizar por parte del administrador de la configuración la solicitud de cambio “Cerrada", en caso de detectarse alguna no conformidad se reasigna nuevamente la implementación al equipo de proyecto.

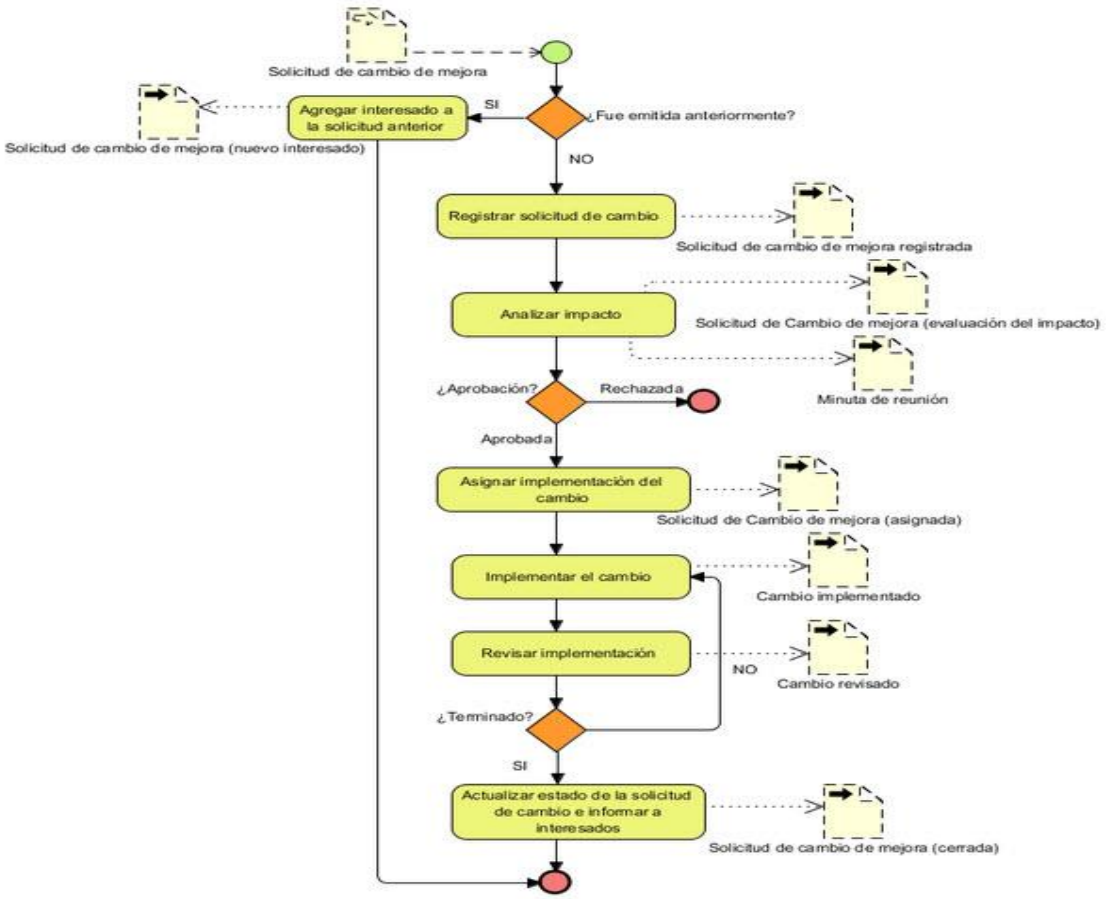


Figura 3: Diagrama del Control de Cambios de CMMI

Con el uso de este modelo, se han obtenido resultados satisfactorios en la actividad productiva de la universidad, y ha demostrado ser uno de los modelos más factibles para lograr la mayor calidad en el proceso de software, por lo que se propone, basar la implementación en este subproceso, siguiendo al detalle cada una de sus actividades. También con el uso de este modelo se ahorrará tiempo en la investigación, pues, en el centro se cuenta con personal calificado al cual acudir, además de existir bibliografía en la cual auxiliarse.

1.5 HERRAMIENTAS PARA EL CONTROL DE CAMBIOS

Se realizará un análisis acerca de algunas de las herramientas que soportan el control de cambios. Para ello se tendrá en cuenta una serie de funcionalidades, las cuales son requeridas para el sistema en cuestión. A continuación, se desglosan dichas funcionalidades y características:

- Gestión de solicitud y aprobación de cambio.
- Registro de los cambios.
- Ajuste al subproceso de control de cambios de CMMI.
- Permitir ver el impacto de los cambios.
- Notificación automática del cambio a través del correo electrónico.

Remedy change management

Esta herramienta proporciona normas, gestión de procesos y capacidad de planificación para incrementar la velocidad y coherencia en la implantación de los cambios y, al mismo tiempo, minimizar los riesgos para el negocio (BMC Software 2007). Desde la solicitud del cambio hasta su verificación, pasando por su planificación y aplicación, Remedy Change Management ayuda a evaluar el impacto, los riesgos y los recursos asociados a los cambios. Además permite la notificación de los cambios a través del correo electrónico. Algunas transiciones de estado en el ciclo de vida RFC sólo pueden ser realizadas por los usuarios que se les ha asignado una función específica. Las diversas etapas (estados) en el ciclo de vida de una solicitud de cambio son (EHEALTH 2015):

Tabla 1: Ciclo de vida de remedy change management

Cambio de estado	Descripción
Abrir proyecto	Se introducen los datos iniciales del proyecto y todavía no es enviada la solicitud de cambio
Autorización de la solicitud	La solicitud está en fase de aprobación esperando la opinión del grupo administración del cambio
Solicitud de cambio	La solicitud está en fase de aprobación para el negocio
Planificación	Las tareas se crean y se asignan a las actividades de planificación
Programado para su aprobación	La solicitud está en fase de aprobación para el administrador del comité del control de cambios
Programado	La solicitud ha sido asignada para su realización.
Implementación en curso	Se está implementando la solicitud
Pendiente	El trabajo con la solicitud se ha suspendido temporalmente
Rechazado	La solicitud ha sido rechazada por el aprobador
Completada	Se ha implementado la solicitud del cambio
Cerrada	La solicitud ya no puede ser modificada
Cancelada	Ya no es necesaria la solicitud del cambio.

En mi opinión, remedy change management sería una buena opción para llevar a cabo el control de los cambios, pues cumple con cada funcionalidad descrita, y aunque su ciclo de vida no es exactamente igual al subproceso llevado a cabo en la universidad, sí es cómodamente adaptable. Su inconveniente, es software propietario, lo que provoca que este fuera del alcance de la universidad.

Sablime

El Sistema de Gestión de la Configuración Sablime es una herramienta perteneciente a la familia del software propietario, potente y eficaz que proporciona versión integrada de control y gestión del cambio de sus elementos de software tales como archivos de código fuente y documentación. Ofrece una solución para rastrear cambios en el software y la documentación para dar mantenimiento, soporte y entrega (TECHNOLOGIES 2006).

Los miembros del equipo o clientes proponen cambios mediante la creación de una nueva solicitud. Cuando se revisan las solicitudes pueden ser asignadas para el estudio o aceptadas para su aplicación. Las solicitudes de cambio son la unidad de control de cambios. Sablime realiza el seguimiento de los cambios que se hacen de cada solicitud y asegura que se entreguen estos cambios (*Sablime Configuration Management System Overview* 2013).

Cada solicitud es asignada a uno o más miembros del proyecto con la debida prioridad y fecha de vencimiento. Si una solicitud requiere gran esfuerzo o se extiende por varias zonas de responsabilidad, se puede subdividir en partes más pequeñas y asignarla de forma independiente, y darle seguimiento en diferentes productos genéricos (*Sablime Configuration Management System Overview* 2013).

Cada solicitud pasa a través de su ciclo de vida y se notifica a los miembros de los eventos vía e-mail. Esta comunicación es clara y promueve la responsabilidad. En cada etapa, Sablime realiza un seguimiento de quién, cuándo y por qué cada acción será tomada.

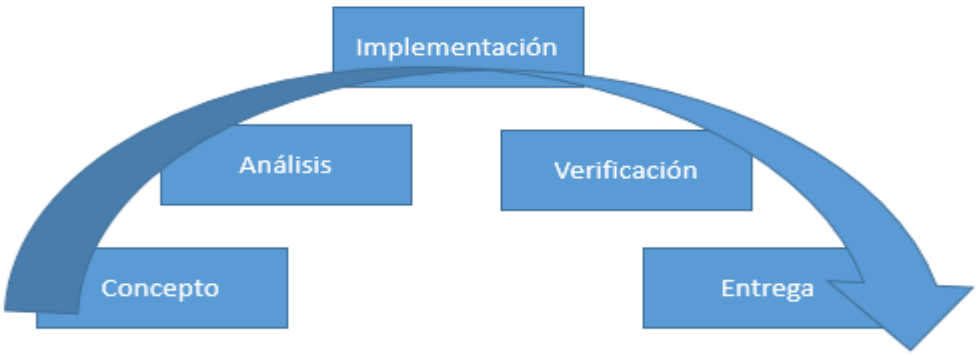


Figura 4: Ciclo de vida de Sablime

La herramienta a pesar de contar con características muy potentes, no se adecua en su totalidad al proceso que se requiere en la universidad, no especifica si lleva un análisis de impacto de las solicitudes y en caso de hacerlo no se describe como lo lleva a cabo. Además posee el inconveniente de ser software propietario.

Serena Dimensions CM

Las dimensiones CM viene para realizar la orquestación y gestión de cambios en el software, alineado con las prácticas de gestión de los cambios de manera eficiente y eficaz. Es posible diseñar e implementar un proceso único con capacidad para gestionar la vida de los tiempos en la aplicación, proporcionando la trazabilidad y auditorías de registros, conectando a los principales repositorios de datos. Serena Dimensions CM maneja cambios de las aplicaciones a través de todas sus plataformas, entornos y equipos (Serena Dimensions 2014).

Los cambios solicitados, se integran con la gestión de la configuración a través de un único modelo de procesos y un repositorio unificado. Proporciona el análisis de impacto de las aplicaciones y aporta una elevada funcionalidad y seguridad al desarrollador, pues visualiza y gestiona el estado de sus solicitudes de desarrollo utilizando una interfaz interactiva que comunica fácilmente el estado entre los miembros del equipo. Monitorear el progreso, identifica cuellos de botella y gestiona el movimiento de las solicitudes entre los propietarios de manera eficiente. Todo ello apoyado por procesos automatizados e incorporando las mejores prácticas de la industria tales como CMMI (Serena Dimensions CM 2016).

A pesar de ser una excelente opción para llevar a cabo el control de los cambios y cumplir al menos parcialmente con cada funcionalidad requerida, su gran obstáculo, al igual que las herramientas anteriores, es software propietario, pues pertenece a Serena Software, una compañía líder en soluciones de tecnología con atención global. Actualmente cuenta con más de 2.500 clientes empresariales, siendo el mayor distribuidor de *Application Lifecycle Management* (ALM – Gestión del Ciclo de Vida de Aplicaciones) independientes en el mundo (Serena Dimensions CM 2016).

20s Change Coordinator

20s Change Coordinator, es un producto que permite gestionar el proceso de solicitudes de cambios de software y realizar seguimiento de defectos. Construye un proceso de gestión de solicitudes de cambio de software específico para el usuario y establece valores aceptables de entrada asociados con las solicitudes de cambio. Mantiene un hilo de comunicación por solicitud, permitiendo archivar registros de solicitudes de cambio según se necesite y generar informes impresos por cada solicitud (Bug Tracking and Defect Tracking Resource 2007). El proceso de control del cambio perteneciente a la herramienta se describe en la Figura 5: Proceso de control de cambio de 20s Change Coordinator.

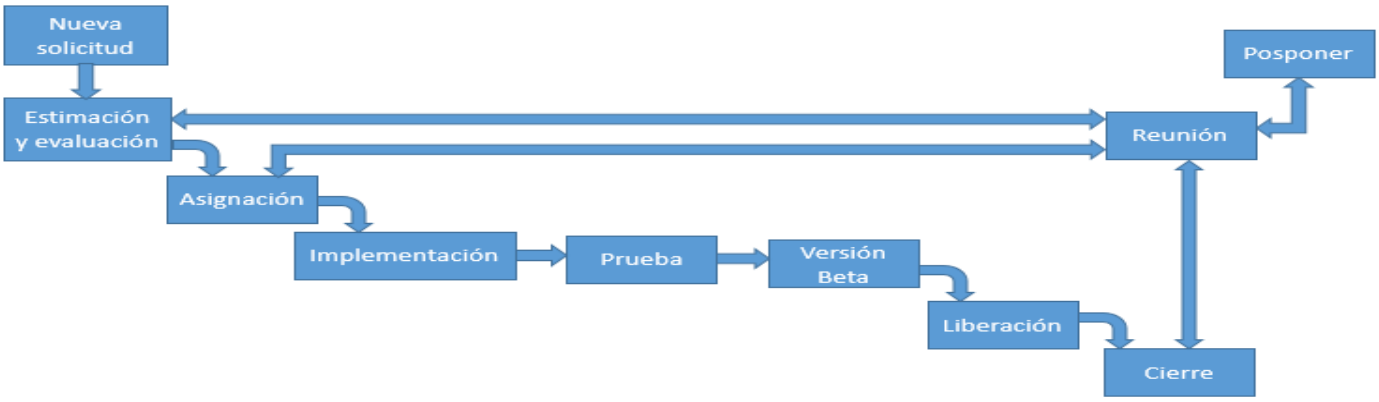


Figura 5: Proceso de control de cambio de 20s Change Coordinator

Como se aprecia, el proceso no está muy lejos de lo esperado, pero con respecto al análisis de impacto no hay suficiente información para describirlo, como también se desconoce cómo es su proceso de notificación. Destacar además que es software propietario, lo cual hace de esta característica su principal agravante para su utilización en la universidad.

Herramienta de Gestión de Cambios MATCHPOINT

Esta herramienta garantiza una segura y eficiente gestión de todos los cambios del software. Apoya completamente los procesos de desarrollo y despliegue, desde la creación de una Solicitud de Cambio hasta el despliegue automatizado en el entorno de producción. MATCHPOINT permite conocer el estado y las actividades para cada Solicitud de Cambio. Las Solicitudes de Cambio registran datos como el estado, la responsabilidad, fecha tope, y prioridad. Es también posible adjuntar documentos u otros objetos a una Solicitud de Cambio, lo que facilita la documentación.

Las Solicitudes de Cambio pueden ser filtradas y organizadas, dando a cada miembro del proyecto la vista que necesita de los datos. Para las Solicitudes de Cambio que provocan un cambio en la aplicación, los objetos modificados o nuevos pueden ser adjuntados a la Solicitud de Cambio. Los objetos adjuntos pueden ser desplegados automáticamente al entorno requerido (Bug Tracking and Defect Tracking Resource 2007).

Algunas de las funcionalidades que ofrece son: notificación por correo electrónico (los usuarios involucrados pueden ser notificados de nuevas Solicitudes de Cambio, o del cambio de estado de alguna

de ellas; dependencia entre Solicitudes de Cambio (si una Solicitud de Cambio depende de otra, esta dependencia puede ser ingresada); documentos adjuntos (para facilitar la documentación, pueden ser adjuntados a la Solicitud de Cambio, conceptos, capturas de pantalla, etc.); registro completo de todas las acciones (cada cambio, manual o automático es registrado en un archivo, lo que permite tener trazas de todas las acciones durante el proceso de Gestión de Cambios) (Bug Tracking and Defect Tracking Resource 2007).

Otra buena opción sin duda alguna es Matchpoint, pero basta recordar que es software propietario, para darse cuenta de su indisposición para la universidad. Además, no está claro que la herramienta realice un análisis del impacto de cada solicitud como se pretende con la correspondiente investigación, y se desconoce además cómo funciona el proceso de control de Cambios.

La siguiente tabla, es un resumen del análisis realizado sobre cada herramienta acerca de su desempeño para llevar a cabo el control de los cambios.

Tabla 2: Resumen del análisis de las herramientas para el control de cambios

	Gestión de solicitud y aprobación de cambio	**Registro de los cambios**	**Ajustable al subproceso de control de cambios implementado en la UCI**	**Envío de notificación de los cambios vía e-mail**	**Permitir ver el impacto de los cambio**	**Licencia gratuita**
Sablime	x	x	x	x	x	
Serena Dimensions CM	x	x	x		x	
Matchpoint	x	x		x		
Change Coordinator	x	x	x			
remedy change management	x	x	x	x	x	

Se puede concluir que emplear estos sistemas en la universidad no sería posible, pues son propietarios. Por otra parte, poseen características en su mayoría muy potentes, las cuales podrían aprovecharse para el desarrollo del nuevo sistema de forma que se provea respuesta a cada funcionalidad requerida. Al no contar con una herramienta capaz de satisfacer las funciones antes descritas en su totalidad, es necesario la creación de un sistema de control de cambios para los proyectos de CEGEL.

1.6 SISTEMA DE CONTROL DE CAMBIOS EN CEGEL

Como parte del trabajo investigativo se aplicó una entrevista a más del 90% de los jefes de proyecto pertenecientes a CEGEL y a sus respectivos administradores de base de datos. La tabla 3 que se muestra en los anexos recoge los datos más significativos a tener en cuenta a la hora de realizar un control de cambios a las bases de datos.

En los resultados de la entrevista se aprecia que llevar a cabo un sistema que controle el versionado en las bases de datos es vital para un trabajo más eficiente. Entre las irregularidades más relevantes se encuentran las siguientes:

En el 100% de los casos los cambios pasan por un proceso de aprobación, pero no se realizan las actividades que propone CMMI, pues no existe un sistema capaz de guiarlos en un proceso organizado y bien definido.

En el 100% de los casos los cambios son notificados de forma verbal, es decir, dependen del factor humano, lo cual puede provocar errores durante el proceso de desarrollo.

En el 100% de los casos no existe un mecanismo que lleve una trazabilidad de los cambios que son implementados sobre un elemento de configuración específico, lo que puede provocar grandes errores durante todo el período de trabajo.

Descritas las irregularidades anteriores se llega a la conclusión que es necesario implementar un sistema capaz de mitigar cada una de estas anomalías.

1.7 CALIDAD EN EL PROCESO DE IMPLEMENTACIÓN DE SOFTWARE

Para lograr la calidad del software es necesario cumplir un conjunto de actividades para ayudar a asegurar todo producto de trabajo de ingeniería del software, como realizar actividades de control y aseguramiento de la calidad en cada proyecto de software, usar métricas para desarrollar estrategias que mejoren el proceso de software y, como consecuencia, la calidad del producto final (PRESSMAN 2005).

A continuación, se desglosan los diferentes conceptos de calidad de software con que se estará trabajando:

El grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario (IEEE, Std. 610, 1990).

Conjunto de rasgos y características de un producto o servicio que le confieren su aptitud para satisfacer necesidades explícitas o implícitas (ISO 9126, 1991).

"Concordancia del software producido con los requisitos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requisitos implícitos no establecidos formalmente, que desea el usuario" (Pressman, 1998).

Para la calidad del software, se tienen dos aristas, una sería construir el producto correcto, enfocada a la calidad del producto. Una segunda arista sería construir correctamente el producto, enfocada a la calidad del proceso. Siguiendo el último enfoque mencionado, se toma como concepto el redactado por la IEEE, pues está más enfocado a la calidad del proceso.

Para validar la calidad del proceso de implementación de software han quedado identificadas las Políticas de IPP-3560:2014 pertenecientes al área de administración y configuración que define CMMI en el programa de mejora.

1.8 PATRONES ARQUITECTÓNICOS

Los patrones arquitectónicos se utilizan para expresar una estructura de organización base o esquema para un software. Proporcionan un conjunto de subsistemas predefinidos, especificando sus responsabilidades, reglas, directrices que determinan la organización, comunicación, interacción y relaciones entre ellos.

Los patrones arquitectónicos heredan mucha de la terminología y conceptos de patrones de diseño, pero se centran en proporcionar modelos y métodos reutilizables específicamente para la arquitectura general de los sistemas de información. En otras palabras quiere decir que a diferencia de los patrones de diseño estas son plantillas incompletas y no se pueden aplicar directamente al código con modificaciones meramente contextuales. Los patrones arquitectónicos a su vez se salen del código puro de la aplicación y suben e incluyen software, hardware, redes, inclusos las personas. El patrón Modelo-Vista-Controlador que se explicará posteriormente es un ejemplo de estos.

Modelo-Vista Controlador

El patrón modelo-vista-controlador separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, e-mail). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

1.9 PATRONES DE DISEÑO

La calidad de diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos. En los patrones GRASP se codifican algunos de ellos, que se aplican al preparar los diagramas de interacción, cuando se asignan las responsabilidades o durante ambas actividades(LARMAN 1999).

Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades. Durante el diseño orientado a objetos, cuando se definen las interacciones entre los objetos, tomamos decisiones sobre la asignación de responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y se nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones (LARMAN 1999). Es por ello que es necesario entender y aplicar varios de los patrones de diseño que se hará mención posteriormente.

Experto

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información

que poseen. El patrón Experto da origen a diseño donde el objeto de software realiza las operaciones que normalmente se aplican a la cosa real que representa. Con este patrón se logra:

- Conservar el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. (Bajo Acoplamiento es un patrón GRASP que examinaremos más adelante.)

- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase, sencillas y más cohesivas que son más fáciles de comprender y de mantener (LARMAN 1999).

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. El patrón Creador indica que la clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear la cosa contenida o registrada. Desde luego, se trata tan solo de una directriz.

Se brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que nos llevaron a elegirla como el parámetro adecuado.

1.10 MÉTRICAS PARA EL DISEÑO

Las métricas referentes al tamaño para las clases orientadas a objetos (OO) se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema OO como un todo. El tamaño general de una clase puede medirse determinando las siguientes medidas: El total de operaciones (heredadas y privadas de la instancia), que se encapsulan dentro de la clase. El número de atributos (heredados y privados de la instancia), encapsulados por la clase.

Estos dos valores son sumados de acuerdo a la clase que se analiza y los resultados son tomados como Cantidad de Procedimientos (CP) que luego son comparados para determinar el Tamaño Operacional de Clases de cada clase (FORNARIS 2009).

Tabla 3: Clasificación de valores

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

Tamaño Operacional de Clase (TOC)

Consiste en medir el tamaño general de una clase tomando el valor del (CP) que están encapsuladas dentro de dicha clase y comparándolo con el promedio donde, Promedio = $(\sum_{n=0}^{i} CP)/i$, donde i = cantidad de clases.

Si el resultado obtenido indica valores grandes, significa que la clase posee un alto grado de responsabilidad, debido a esto se reducirá la reutilización, se hará mucho más difícil la implementación y la realización de pruebas de dicha clase. Por tanto, mientras menor sea el valor para el TOC se hará más fácil la reutilización de dicha clase dentro del sistema (LINCKE 2009).

Tabla 4: Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica TOC.

Clasificación	Categoría	Criterio
Responsabilidad	Baja	CP <=Promedio
	Media	Promedio<= CP <=2*Promedio
	Alta	CP >2*Promedio
Complejidad de implementación	Baja	CP <=Promedio
	Media	Promedio<= CP <=2*Promedio
	Alta	CP >2*Promedio
Reutilización	Baja	CP >2*Promedio
	Media	Promedio<= CP <=2*Promedio
	Alta	CP <= Promedio

Relaciones entre Clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad (*Desarrollo web* 2013), donde Promedio = $(\sum_{n=0}^{i} RC)/i$, donde i = cantidad de clases.

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

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

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

Tabla 5: Rango de valores para la evaluación de los atributos de calidad relacionados con la métrica RC

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Baja	1
	Media	2
	Alta	Cantidad de relaciones de uso > 2
Complejidad de mantenimiento	Baja	Cantidad de relaciones de uso <= Promedio
	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio
Cantidad de pruebas	Baja	Cantidad de relaciones de uso <= Promedio
	Media	Promedio <= Cantidad de relaciones de uso <= 2* Promedio
	Alta	Cantidad de relaciones de uso > 2* Promedio

1.11 TECNOLOGÍAS Y HERRAMIENTAS A UTILIZAR

CMMI

Los modelos CMMI (*Capability Maturity Model® Integration*) son colecciones de buenas prácticas que ayudan a las organizaciones a mejorar sus procesos. Estos modelos son desarrollados por equipos de producto con miembros procedentes de la industria, del gobierno y del *Software Engineering Institute* (SEI). Este modelo, denominado CMMI para Desarrollo (CMMI-DEV), proporciona un conjunto completo e integrado de guías para desarrollar productos y servicios.

Una de las constelaciones que define CMMI es la versión para desarrollo (CMMI-DEV). La misma es un modelo de referencia que cubre las actividades para desarrollar tanto productos como servicios. Proporciona una orientación para aplicar las buenas prácticas CMMI en una organización de desarrollo. Las buenas prácticas del modelo se centran en las actividades para desarrollar productos y servicios de calidad con el fin de cumplir las necesidades de clientes y usuarios finales.

Metodología de desarrollo para la Actividad productiva de la UCI

El Proceso Unificado Ágil de *Scott Ambler* o *Agile Unified Process* (AUP), describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo:

- Desarrollo Dirigido por Pruebas (test driven development - TDD en inglés).
- Modelado ágil.
- Gestión de cambios ágil.
- Refactorización de base de datos para mejorar la productividad.

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva (SÁNCHEZ 2014).

Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas, por lo que se incorpora a AUP el Modelo

CMMI-DEV v1.3, el cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad.

Esta metodología está compuesta por 7 disciplinas, los flujos de trabajos: Modelado de negocio, Requisitos y Análisis, y diseño en AUP, que están unidos en la disciplina Modelo, y la disciplina implementación. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión se cubren con las áreas de procesos que define CMMI- DEV v1.3 para el nivel 2, serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto).

Tiene cuatro escenarios para modelar el sistema en los proyectos:

- Proyectos que modelen el negocio con Casos de Uso del Negocio (CUN) solo pueden modelar el sistema con Casos de Uso del Sistema (CUS).
- Proyectos que modelen el negocio con Modelo Conceptual (MC) solo pueden modelar el sistema con CUS.
- Proyectos que modelen el negocio con Descripción de Proceso de Negocio (DPN) solo pueden modelar el sistema con Descripción de requisitos por proceso (DRP).
- Proyectos que no modelen negocio solo pueden modelar el sistema con Historias de Usuario (HU) (SÁNCHEZ 2014).

PHP

Acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. Al nivel más básico, PHP permite procesar la información de formularios, y generar páginas con contenidos dinámicos. Una de sus características más potentes y destacables es su soporte para una gran cantidad de bases de datos, pues escribir una interfaz vía web para una base de datos es una tarea simple con PHP. PHP también soporta el uso de otros servicios que usen protocolos como IMAP, SNMP, NNTP, POP3, HTTP y derivados (BAKKEN *et al.* 2001).

Las razones de utilizar este lenguaje se deben a su poder y sencillez. PHP está disponible para la mayoría de sistemas operativos existentes. Desde Unix, Linux, Microsoft Windows, MAC, entre otros. En cuanto a su costo, es un software libre, por lo que puede ser utilizado sin problemas. Su rendimiento es muy bueno

y verdaderamente eficiente, pues utilizando un servidor modesto se puede atender millones de peticiones al día, lo cual es más que suficiente para el desarrollo de la aplicación en cuestión. Con PHP se tiene acceso al código fuente, es decir si se desea agregar o modificar algo para obtener un funcionamiento de acuerdo a ciertas necesidades se puede hacer con total libertad.

PHP es uno de los lenguajes para web más populares y con el que se está más cómodo trabajando, pues ya se ha trabajado con el mismo en trabajos propuestos en la asignatura de PID, además de haberse recibido clases durante todo un semestre sobre el tema, logrando desarrollar un proyecto final de la asignatura de Programación 5(P5). Además, la versión 5 de PHP, la cual es la que será utilizada, está diseñada para soportar características de programación orientada a objetos, lo cual es muy favorable pues se ha trabajado con este soporte a todo lo largo de la carrera.

JQuery

JQuery es un framework para el lenguaje Javascript que brinda muchas facilidades a la hora de programar. JQuery implementa una serie de clases (de programación orientada a objetos) que permite programar sin preocuparse del navegador con el que se está visitando el usuario y funcionan de exacta forma en todas las plataformas más habituales. Ofrece una infraestructura con la que se tiene mucha mayor facilidad para la creación de aplicaciones complejas del lado del cliente, por ejemplo, gracias a su perfecta integración con Ajax, se obtiene una ayuda en la creación de interfaces de usuario, efectos dinámicos, entre otras, siendo esta integración unos de los motivos principales para su uso.

Todas estas ventajas, con jQuery se obtienen de manera gratuita, ya que el marco de trabajo tiene licencia para uso en cualquier tipo de plataforma, personal o comercial, siendo esto otra razón para utilizarlo. Es un producto profesional, estable, bien documentado y con un gran equipo de desarrolladores a cargo de su mejora y actualización, pues existe una dilatada comunidad de creadores de complemento o componentes, lo que hace fácil encontrar soluciones ya creadas en jQuery para implementar asuntos como interfaces de usuario, galerías, votaciones, efectos diversos, etc. Estos ejecutables son sinónimos de ahorro en tiempo y esfuerzo.

La UCI, actualmente tiene muchos proyectos que hacen uso de JQuery, pues es fácil de investigar y de poner en marcha en la aplicación. Además, el equipo de desarrollo tiene experiencia usándolo, pues fue

utilizado en tareas años anteriores. Es un producto con muy buena aceptación por parte de los programadores en todo el mundo y con un grado de penetración en el mercado muy amplio.

PostgreSQL

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL, es uno de los gestores de bases de datos de código abierto más avanzado hoy en día, ofrece control de concurrencia multiversión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, Tcl y Python) (*Bazaar* 2009).

Es un sistema de base de datos fácil de aprender, altamente configurable, con lo cual puede personalizarse de acuerdo al escenario donde será utilizado. Es multiplataforma (cuenta con versiones para sistemas operativos Unix-like y Windows), y es extensible pues pueden implementar nuevos tipos de datos, funciones, operadores, y lenguajes e incorpora un sistema de recolección de estadísticas en tiempo real de transacciones.

Después de dar a conocer algunas de las principales características de este potente gestor, no hay motivo para excluirlo de su uso en el sistema a desarrollar. En múltiples asignaturas durante la carrera, el gestor utilizado fue PostgreSQL, por lo que se decide su uso para la correspondiente aplicación, con el objetivo de minimizar el tiempo de desarrollo del sistema.

Apache

Es un servidor web HTTP, para plataformas Unix, Microsoft Windows, Macintosh y otras. La arquitectura del servidor Apache es modular. El servidor consta de diversos módulos que aportan muchas funcionalidades que podría considerarse básica para un servidor web. Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, así como otras funciones que lo califican como un servidor robusto y rápido de código abierto (MESTRAS 2013).

CodeIgniter

Es un conjunto de herramientas para personas que construyen aplicaciones web usando PHP. Su objetivo es desarrollar proyectos mucho más rápido de lo que podría si escribiese el código desde cero, proveyendo un rico juego de librerías para tareas comúnmente necesarias y una interfaz simple y estructura lógica para acceder a las bibliotecas. CodeIgniter permite creativamente enfocarse en el proyecto minimizando la cantidad de código necesario para una tarea (LOZANO 2012).

CodeIgniter usa el acercamiento Modelo-Vista-Controlador, permitiendo una buena separación entre lógica y presentación. Decir también que es libre, pues se encuentra bajo una licencia de código abierto Apache/BSD-style. Es verdaderamente liviano y el núcleo del sistema sólo requiere unas pocas y pequeñas bibliotecas, contrastando con muchos entornos de trabajo que requieren significativamente más recursos. Las librerías adicionales son cargadas dinámicamente a pedido, basado en sus necesidades para un proceso dado, provocando un sistema base delgada y bastante rápida.

Bootstrap

Es un marco de trabajo que simplifica el proceso de creación de diseños web combinando CSS y JavaScript. Las mayores ventajas es que podemos crear interfaces que se adapten a los distintos navegadores, se integra perfectamente con las principales librerías Java Script, por ejemplo JQuery además de contar con numerosos componentes webs que nos ahorrarán mucho esfuerzo y tiempo (MESTRAS 2014).

Visual Paradigm

Es una herramienta CASE de modelado UML que proporciona un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Es capaz de soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de diferentes tipos de diagramas.

Esta herramienta está disponible en múltiples plataformas (Windows, GNU/Linux) y es fácil de instalar y actualizar. Es distribuida bajo licencia libre, tiene capacidad de ingeniería directa e inversa, soporta UML y ORM, con versiones son compatibles entre sí. Brinda funcionalidades para la generación de código fuente a partir de diagramas de clases.

Está diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque orientado a objetos, enfoque el cual se ha estudiado a lo largo de toda la carrera.

1.12 CONCLUSIONES PARCIALES

Después del desarrollo del anterior capítulo se arribaron a las siguientes conclusiones:

- Se trató lo referente a conceptos como el de Gestión de la configuración, control de cambios y calidad en el proceso de implementación de software, los cuales crearon la base teórica de la investigación.
- Se analizaron un conjunto de herramientas existentes en la actualidad para el control de los cambios, dando como conclusión que ninguna cumple en su totalidad con las funcionalidades requeridas por la universidad.
- Se plasmaron resultados de una entrevista realizada a cada jefe y administrador de bases de datos de los proyectos de CEGEL, demostrando la necesidad de una herramienta que responda a las deficiencias obtenidas.
- Se describieron los patrones y métricas del diseño a tener en cuenta para la realización de la aplicación, garantizando una mayor calidad en el diseño.
- Se realizó un estudio de las herramientas, lenguajes y tecnologías más importantes en la actualidad, quedando planteadas las seleccionadas para el desarrollo de la herramienta a confeccionar.

Capítulo 2. Arquitectura y diseño del sistema

2.1 INTRODUCCIÓN

En el presente capítulo se realiza la descripción de la solución propuesta para el sistema de control de cambios durante el proceso de desarrollo de software. Se listan, describen, y validan los requisitos del sistema y se define una arquitectura para el mismo, realizando un modelo de diseño a través de diagramas de clases y despliegue. Además se especifica el estándar de codificación empleado en cada clase implementada, de manera que se muestre el código de la forma más clara posible.

2.2 ESPECIFICACIÓN DE LOS REQUISITOS DE SOFTWARE

Un requisito de software podría definirse como una condición o capacidad que debe encontrarse o estar en un sistema o componente para satisfacer un contrato, norma, especificación u otro documento impuesto formalmente. El conjunto de todas las necesidades es el fundamento para el consiguiente desarrollo del sistema o componente (IEEE. Glosario de Terminología Informática 2009).

Los requisitos que debe satisfacer la propuesta de solución se identificaron de conjunto con el cliente y siguiendo las prácticas actuales de desarrollo de software analizadas en el capítulo anterior. Se agruparon en requisitos funcionales (RF) y no funcionales (RNF), según el módulo de trabajo al que pertenecen.

Requisitos funcionales

Módulo de Administración y Configuración:

Tabla 6: Requisitos funcionales Gestionar usuario

Gestionar usuario		
RF	Complejidad de implementación	Prioridad para el cliente
RF 1. Registrar usuario	Media	Media
RF 2. Modificar usuario	Media	Media
RF 3. Desactivar usuario	Media	Media

RF 4. Buscar usuario	Media	Media

Tabla 7: Requisitos funcionales Gestionar Roles

Gestionar Roles		
RF	Complejidad de implementación	Prioridad para el cliente
RF 5. Agregar rol	Media	Alta
RF 6. Modificar rol	Media	Alta
RF 7. Eliminar roles	Media	Alta

Tabla 8: Requisitos funcionales Gestionar proyecto

Gestionar proyecto		
RF	Complejidad de implementación	Prioridad para el cliente
RF 8. Abrir proyecto	Alta	Alta
RF 9. Modificar proyecto	Media	Alta
RF 10. Definir miembros del proyecto	Alta	Alta
RF 11. Cerrar proyecto	Baja	Alta

Módulo de Control de Cambios:

Tabla 9: Requisitos funcionales del módulo control de cambio

RF	Complejidad de implementación	Prioridad para el cliente
RF 12. Registrar solicitud	Alta	Alta
RF 13. Editar solicitud	Media	Alta
RF 14. Cancelar solicitud	Media	Alta
RF 15. Buscar solicitudes similares a otras solicitudes	Baja	Media
RF 16. Análisis de impacto de la solicitud	Alta	Alta
RF 17. Aprobar solicitud de cambio	Alta	Alta
RF 18. Registrar implementación del cambio	Alta	Alta
RF 19. Emitir notificaciones por correo electrónico	Media	Alta

Módulo de elementos de configuración

Tabla 10: Requisitos funcionales Gestionar elementos de la configuración

Gestionar elementos de la configuración		
RF	Complejidad de implementación	Prioridad para el cliente
RF 20. Adicionar elementos de configuración	Baja	Alta
RF 21. Modificar elementos de configuración	Baja	Alta
RF 22. Eliminar elementos de configuración	Baja	Alta

Tabla 11: Requisitos funcionales Gestionar requisitos

Gestionar requisitos		
RF	Complejidad de implementación	Prioridad para el cliente
RF 23. Adicionar requisitos	Baja	Alta
RF 24. Modificar requisitos	Baja	Alta
RF 25. Eliminar requisitos	Baja	Alta

Requisitos No Funcionales

Requisitos de usabilidad

RNF 1. Utilizar nombres de componentes que sean de lenguaje común para los usuarios.

RNF 2. El sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.

Requisitos de confiabilidad

RNF 3. Cerrar la sección de trabajo si el sistema se encuentra inactivo por más de 5 minutos.

Requisitos de eficiencia

RNF 4. Conectar concurrentemente hasta 100 usuarios al sistema.

Requisitos de mantenimiento

RNF 5. El sistema debe contar con un mantenimiento semestral a su servidor, garantizando la integridad de los datos almacenados en el mismo.

RNF 6. Requisitos mínimos de software

Tener instalado Google Chrome o Mozilla Firefox en su versión v30 en adelante.

RNF 7. Requerimientos mínimos para el servidor:

Ordenador Pentium IV o superior. 2 GB de Memoria RAM o superior. Disco duro de 40 GB o superior.

2.3 VALIDACIÓN DE REQUISITOS

La validación es la actividad de la ingeniería de requisitos que permite demostrar si los requisitos definidos del sistema son los deseados por el cliente y si pueden ser implementados. Es una actividad muy importante pues de ella depende que no existan elevados costos de mantenimiento en el software.

Las listas de chequeo son herramientas de verificación que contienen una lista de factores claves para comprobar e investigar con respecto a la viabilidad de los requisitos de software. Destacar que la lista de chequeo debe ser lo más completa posible en cuanto a los elementos a evaluar y así garantizar que todos los requisitos sigan los estándares de calidad que son definidos en dichas listas de chequeo. A continuación, se enumeran los atributos con una breve descripción de los objetivos que persiguen:

Completo: un requisito está completo si no necesita ampliar detalles para su comprensión y posterior implementación. Debe contener toda la información necesaria para que el desarrollador diseñe y desarrolle esa porción de funcionalidad.

No ambiguo: un requerimiento no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.

Probado: un requisito es probado si y solo si, existe un proceso finito por el cual una máquina o persona puede chequear que el producto de software completa el requisito.

Modificable: cualquier cambio puede realizarse de manera sencilla, completamente y consistentemente manteniendo estructura y estilo.

Único: el requisito debe tener un identificador único.

Consistente: el requisito es congruente con otros requisitos relacionados.

Correcto: cada requisito escrito es uno que el software tiene que cumplir y debe describir con precisión la funcionalidad a construir.

Traceable: el origen de cada requisito está claro y la Especificación de Requisitos de Software (ERS) facilita le referencia a cada requisito en desarrollos futuros o mejoras de documentos.

Factible: debe ser posible la implementación de cada requisito con las capacidades y limitaciones de los recursos disponibles.

Priorizado: asignar una prioridad de implementación a cada requisito para indicar cuan esencial es un requisito en una versión particular del producto.

Necesario: cada requisito debe describir una capacidad o característica que el usuario realmente necesita o que es imprescindible para la conformidad con un sistema externo o estándar. Importante mantener trazabilidad a la fuente.

A los requisitos definidos en esta investigación se le aplicó la lista de chequeo perteneciente al programa de mejora y se obtuvo un resultado positivo donde todos los requisitos podían ser implementados, pues cumplen satisfactoriamente cada uno de los atributos explicados anteriormente.

2.4 ANÁLISIS Y DISEÑO

En todo desarrollo de sistemas de software es de suma importancia el seguir alguna especificación que permita a los desarrolladores tener una disciplina que haga que todas las etapas del desarrollo del sistema, desde la pesquisa inicial de requisitos hasta las pruebas finales del sistema, sean no solo más coherentes sino también más formales.

El desarrollo de software que este sistema propone, al ser una herramienta que pretende tener aplicación dentro del contexto de un problema real, tiene que seguir un proceso de análisis y diseño que proporcione los cimientos bajo los cuales se va desarrollar la aplicación conjuntamente.

Dentro del proceso de análisis es fundamental que, a través de una colección de requisitos funcionales y no funcionales, el desarrollador o desarrolladores del software comprendan completamente la naturaleza de

los programas que deben construirse para desarrollar la aplicación, mientras mediante el diseño se traducen requisitos en una representación de software.

2.5 DESCRIPCIÓN DE REQUISITOS FUNCIONALES

Se realizó la descripción a cada uno de los 25 requisitos funcionales existentes para la confección del sistema. A continuación, en la tabla 2 se realiza la descripción del requisito funcional registrar usuario, el cual se toma como ejemplo debido a su importancia crítica para el sistema.

Tabla 12: Descripción del requisito Registrar Solicitud

<table>
<tr><th>Nº</th><th>Nombre</th><th>Descripción</th><th>Complejidad</th><th>Prioridad para cliente</th></tr>
<tr><td>RF14</td><td>Registrar Solicitud</td><td>El usuario deberá registrar el tipo de solicitud, tipo de elemento, nombre del elemento y una descripción de la solicitud.</td><td>Alta</td><td>Alta</td></tr>
<tr><td></td><td colspan="4">Prototipo</td></tr>
<tr><td></td><td colspan="4"> 
</td></tr>
<tr><td></td><td>Campos</td><td>Tipos de Datos</td><td colspan="2">Reglas o Restricciones</td></tr>
<tr><td></td><td>Descripción de la solicitud</td><td>Cadena de caracteres</td><td colspan="2">Puede contener solo letras y números.</td></tr>
</table>

		No debe contener caracteres extraños, como (*, /, ¨+,-).
Tipo de Solicitud	Cadena de caracteres	Debe seleccionar un tipo de solicitud
Tipo de Elemento	Cadena de caracteres	Debe seleccionar un tipo de elemento.
Observaciones		

2.6 DISEÑO ARQUITECTÓNICO

En el diseño arquitectónico se definen las relaciones entre los principales elementos estructurales del programa. El siguiente diagrama muestra cada una de las clases que contiene la herramienta y se denotan las relaciones entre cada una de ellas:

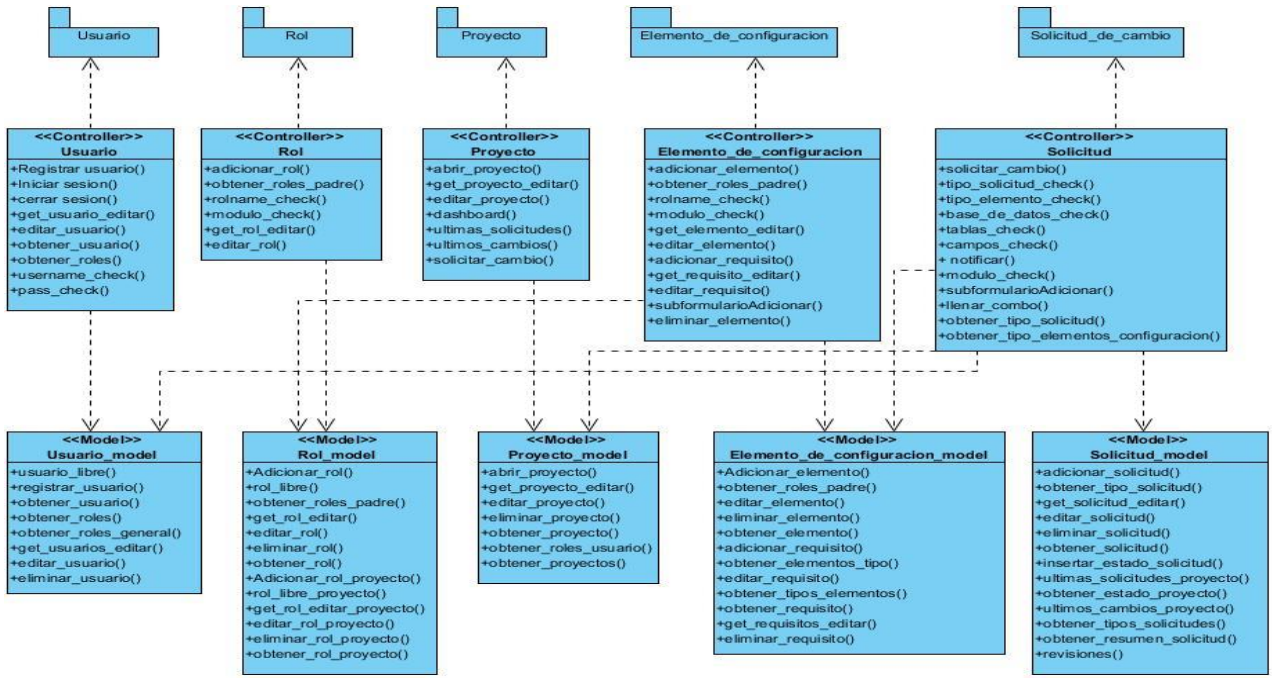


Figura 6: Diagrama de Clases del diseño

En la Figura 6: Diagrama de Clases del diseño, se ejemplifica el uso de diferentes patrones de diseño, los cuales son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Se hizo uso del patrón arquitectónico Modelo-Vista-Controlador, permitiendo que de esta manera los modelos presentes en el diagrama implementen la lógica del negocio y el acceso a datos. Por otra parte las clases controladoras se encargan de realizar las conexiones entre los datos y las vistas, donde estas últimas se encuentran en el paquete usuario como se muestra en la Figura 7.

Figura 7: Patrón Modelo-Vista-Controlador

El diagrama de interacción de la Figura 8, refleja las decisiones en cuanto a la asignación de responsabilidades y el modo en el que deberían interactuar los objetos. Obsérvese la considerable reflexión que se hizo para llegar a este diseño, basada en los patrones GRASP; el diseño de las interacciones entre objetos y la asignación de responsabilidades.

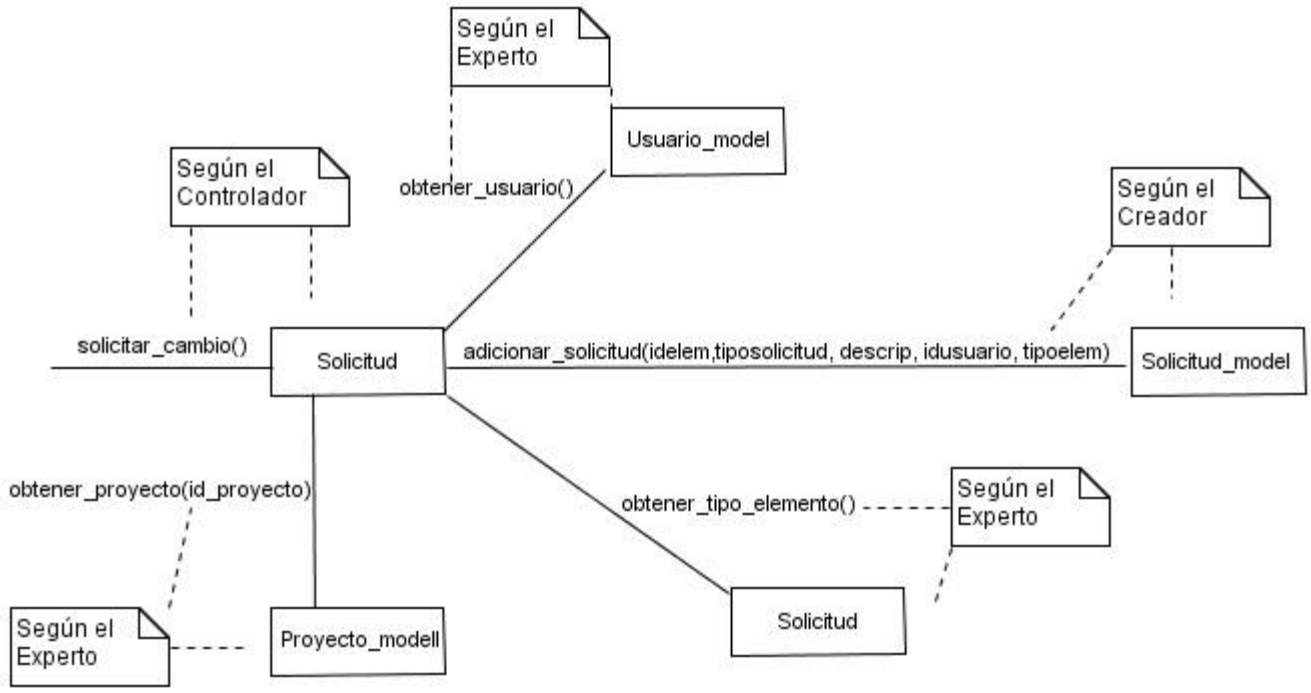


Figura 8: Diagrama de interacción para adicionar solicitud

2.7 MODELO DE DATOS

Un Diagrama o Modelo Entidad Relación (a veces denominado por sus siglas, E-R "Entity Relationship", o, "DER" Diagrama Entidad Relación), es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información, así como sus interrelaciones y propiedades.

Para el propósito actual, un Modelo de Datos no va ser más que un conjunto de conceptos que pueden servir para describir la estructura de una BD, esto se refiere a tipos de datos, sus vínculos y las restricciones que deben cumplir estos datos (ABRAHAM SILBERSCHATZ 2002).

Con el siguiente diagrama Entidad Relación, realizado utilizando la herramienta CASE Visual Paradigm para UML, se logra la creación de una base de datos que permita el almacenamiento de los datos necesarios para el negocio, de forma que garantice la integridad y el rendimiento da la base de datos en el momento de la obtención de cada dato que se maneja.

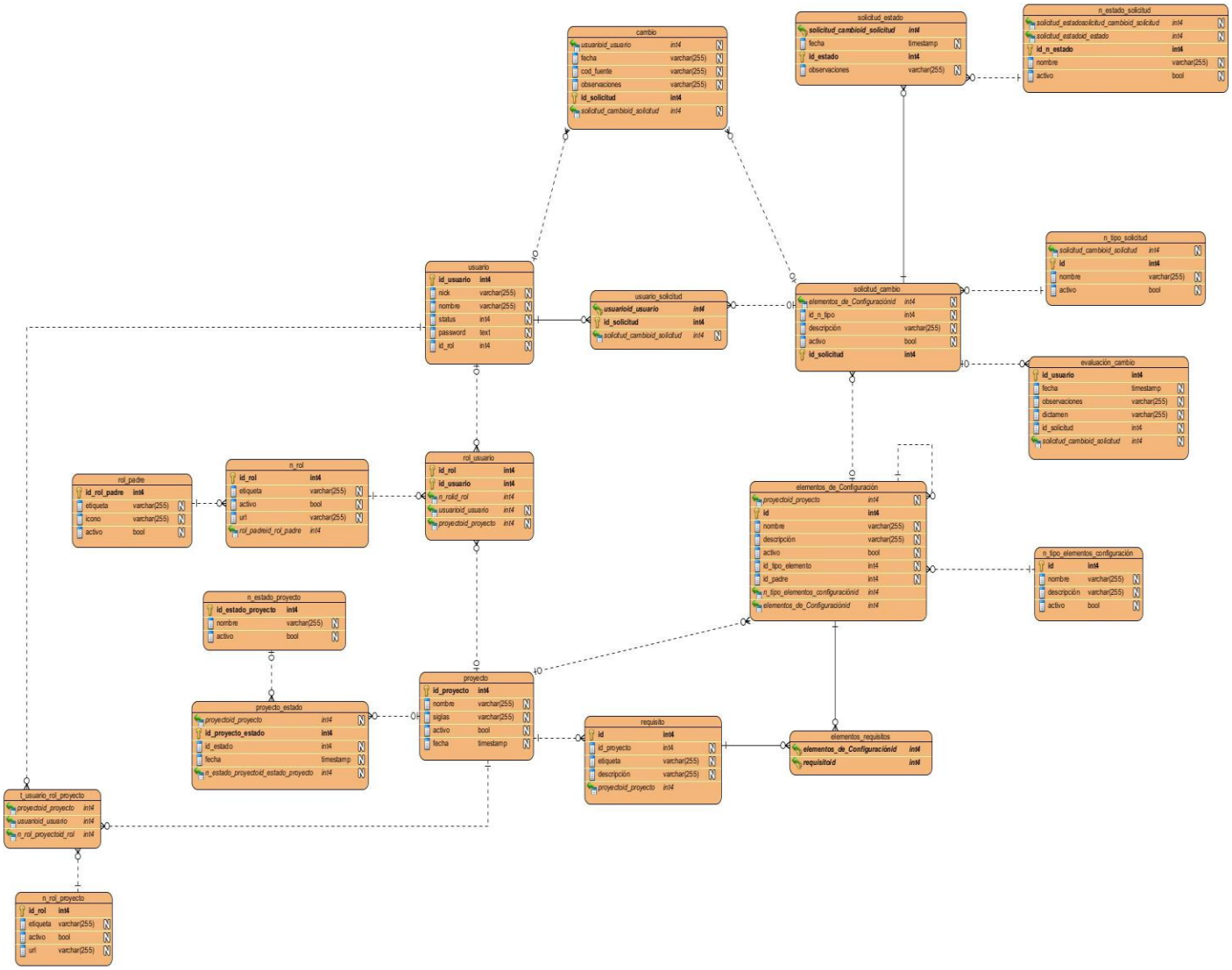


Figura 9: Diagrama Entidad-Relación

En el diseño de las bases de datos y en el modelado de datos en general se presentan elementos repetitivos en disímiles modelos, los que correctamente identificados pasan a ser patrones de diseño. De manera general se puede decir que los patrones constituyen una solución estándar para un problema común, en este caso para el diseño de BD. A continuación, se expondrán los patrones que fueron identificados en el diagrama de la Figura 9.

Árboles simples

Patrón normalmente utilizado cuando el árbol es la representación de una estructura de datos. Los elementos a almacenar son del mismo tipo, es decir, pueden ser almacenados en la misma entidad. No pueden existir ciclos, es decir, un hijo no puede ser su propio padre.

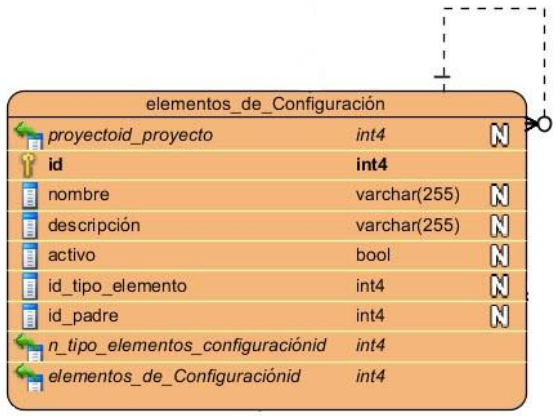


Figura 10: Patrón Árbol Simple

Árboles fuertemente codificados

En este caso a cada nivel del árbol se le asocia una entidad. Normalmente constituyen relaciones de 1 a muchos (n). Se utiliza para representar jerarquías donde es bien conocida la estructura y es importante representar la correspondencia, por ejemplo, las estructuras organizacionales. Es importante señalar que este patrón debe utilizarse sólo en los casos en que los cambios en la estructura a representar sean poco probables. Así como aclarar que el patrón admite tantos niveles como requiera la jerarquía que se vaya a representar.

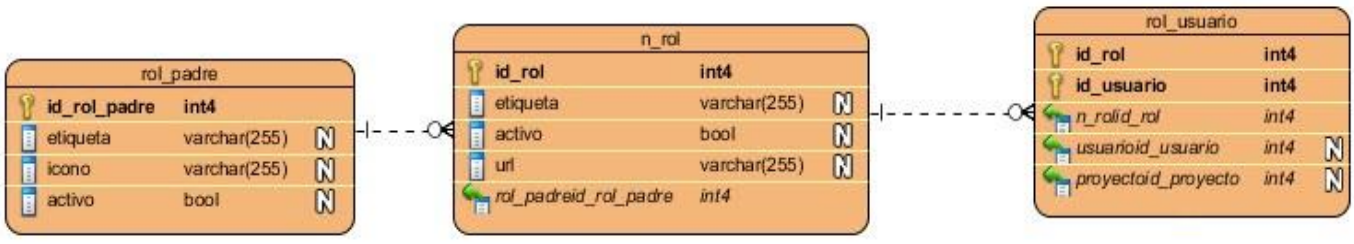


Figura 11: Patrón Árbol Fuertemente Codificado

Patrón de llaves subrogadas

Este patrón es muy utilizado pues facilita la interacción con la BD en un futuro. El mismo plantea que se genere una llave primara única para cada entidad, en vez de usar un atributo identificador en el contexto dado. Esto permite que las tablas sean más fáciles de consultar a partir del identificador, pues todos tienen el mismo tipo en cada una de las tablas.

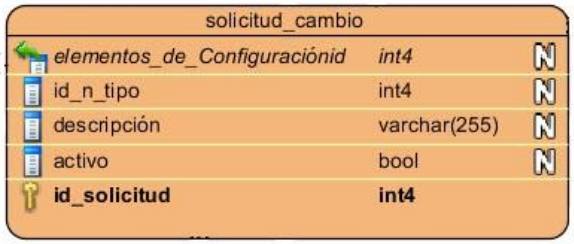


Figura 12: Patrón de llave subrogada

2.8 ESTÁNDARES DE CODIFICACIÓN

Los estándares de codificación son aquellos que permiten entender de manera rápida y sencilla el código empleado en el desarrollo de un software. Garantizan el mantenimiento óptimo de dicho código por parte del programador (CALLEJA 2009). La siguiente tabla muestra los estándares de codificación utilizados en el código de la aplicación.

Tabla 13: Estándares de codificación utilizados

REGLA	DESCRIPCIÓN	Sistema
MÁXIMA LONGITUD DE LÍNEAS	Las líneas de código son limitadas a un número máximo de caracteres.	Limitar todas las líneas de código (excepto los archivos **.js** y .**css**) a un máximo de 80 caracteres.
LÍNEAS EN BLANCO	Separar funcionalidades de alto nivel utilizando líneas en blanco.	Todas las funcionalidades de alto nivel serán separadas por dos líneas en blanco.
ESPACIOS EN BLANCO EN EXPRESIONES Y SENTENCIAS	Evitar el uso innecesario de espacios en blanco en determinadas situaciones.	Se evitará el uso de espacios en blanco en las siguientes situaciones:

		• Inmediatamente dentro de paréntesis, corchetes y llaves. • Inmediatamente antes de una coma, un punto y coma o dos puntos. • Inmediatamente antes del paréntesis que comienza la lista de argumentos de una función.
COMENTARIOS EN EL CÓDIGO FUENTE	Durante el desarrollo de la aplicación resulta útil realizar comentarios explicativos que describan que hace el código.	Los comentarios deben ser oraciones completas. Cada línea de un comentario comienza con el símbolo de numeral (#) y están indendatos al mismo nivel que ese código. Si un comentario es una frase u oración, su primera palabra de comenzar con mayúscula. Si un comentario es corto puede omitirse el punto final.
ESTILOS DE NOMBRAMIENTO	Se distingue los siguientes estilos de nombramiento: • minúscula (lowercase) • minúscula_con_guiones_bajos (lowercase_with_underscores) • MAYÚSCULA (UPPERCASE) • MAYÚSCULA_CON_GUIONES_BAJOS (UPPERCASE_WITH_UNDERSCORES) • PalabrasEnMayúscula (CamelCase) • minúsculaMayúscula (mixedCase)	Para el nombramiento de los métodos y declaración de variables se utilizó el estilo l**owercase**. En caso de ser la palabra compuesta es separada por guiones según el estilo **lowercase_with_underscores**. Además, se hizo empleo de un prefijo único delante de cada nombre capaz de identificar las funciones relacionadas a un mismo método.

Diagrama de despliegue

Mediante el diagrama de despliegue se muestran las relaciones físicas de los distintos nodos que componen el sistema y la repartición de los componentes sobre dichos nodos. A través de este se captura la configuración de los elementos de procedimientos y las conexiones entre estos en el sistema. El modelo de despliegue consta de uno o más nodos, dispositivos y conectores entre nodos, y entre nodos y dispositivos. El diagrama de despliegue que se propone se muestra en la Figura 13: Diagrama de despliegue del sistema.

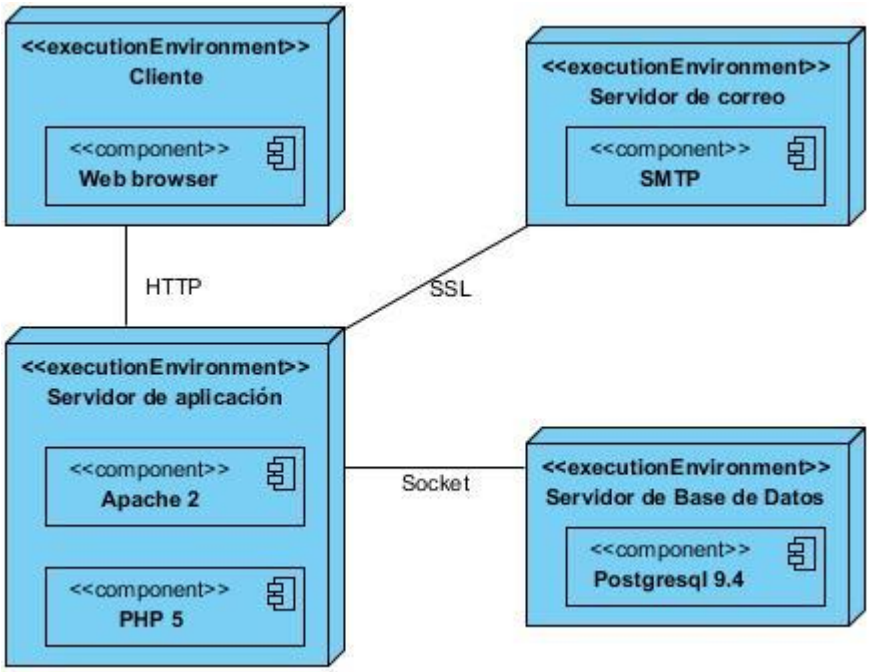


Figura 13: Diagrama de despliegue del sistema

El sistema tendrá una distribución física compuesta por una PC-cliente en la cual estará instalado un navegador web donde se ejecutará la aplicación en su lado cliente. Se contará con un servidor de Base de Datos con la utilización de postgresql 9.4, y con un servidor de aplicación donde se encuentra desplegada la aplicación. El servidor de correo, es empleado para garantizar el envío de notificaciones para cada solicitud que se realice.

2.9 CONCLUSIONES PARCIALES

Después del desarrollo del anterior capítulo se arribaron las siguientes conclusiones:

- Se abordó lo referente a la fase de ejecución donde se definieron los requisitos funcionales, no funcionales y se elaboró una descripción de los mismos, permitiendo así definir las propiedades y bases para el desarrollo del sistema a implementar.
- Se describe la arquitectura y los patrones de diseño y bases de datos para brindar un mayor entendimiento, organización y claridad para la implementación de la solución.
- Queda definido el estándar de codificación a utilizar para garantizar un mayor entendimiento y organización del código.
- Se realiza un diagrama de despliegue permaneciendo claras las relaciones y configuraciones entre los componentes del sistema.

Capítulo 3. Verificación y Validación

3.1 INTRODUCCIÓN

Se hace uso en el capítulo de diferentes métricas para validar el funcionamiento de todos los requisitos identificados y el diseño planteado para el desarrollo del sistema informático. Para la verificación del sistema se realizan pruebas de caja negra examinando las funcionalidades a nivel de interfaces en el sistema y se realiza la validación de las variables que se plantearon en el problema a resolver.

3.2 ESTRATEGIA DE PRUEBA

La estrategia de pruebas que se define para verificar el correcto funcionamiento de la herramienta estará encaminada al nivel de pruebas unitarias y de sistema. Una prueba unitaria es una forma de comprobar el correcto funcionamiento de un módulo de código, con el objetivo de aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Por otra parte, las pruebas de sistema se le realizaran a toda la herramienta como un todo y así comprobar su correcto funcionamiento.

Se utiliza el método de caja negra para saber si un determinado producto, cumple con la funcionalidad para lo que fue creado. Son llevadas a cabo sobre la interfaz del software, es decir, de la función, actuando sobre ella como una caja negra, proporcionando unas entradas y estudiando las salidas para ver si concuerdan con las esperadas.

Se emplea la técnica de prueba partición de equivalencia que asegura la creación de un caso de prueba para cada uno de los requisitos definidos.

3.3 MÉTRICAS PARA VALIDAR EL DISEÑO

Una métrica es una definición operativa que describe, en términos muy específicos, lo que algo es y cómo lo mide el proceso de control de calidad(Guía de los Fundamentos de la Dirección de Proyectos 2004).

A continuación, se ejemplificará el uso de las técnicas usadas para validar el diseño.

Tamaño Operacional de Clases (TOC)

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

- Responsabilidad: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio.
- Complejidad de implementación: grado de dificultad que tiene implementar un diseño de clase determinado.
- Reutilización: consiste en el grado de reutilización presente en una clase o estructura de clases, dentro de un diseño de software.

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

La aplicación del instrumento de evaluación de la métrica TOC para el número total de operaciones fue desarrollada para una muestra de 10 clases, donde se encuentran todas las clases controladoras y todos los modelos, quedando solo sin analizar las clases pertenecientes a las vistas. A continuación, se muestran los resultados obtenidos:

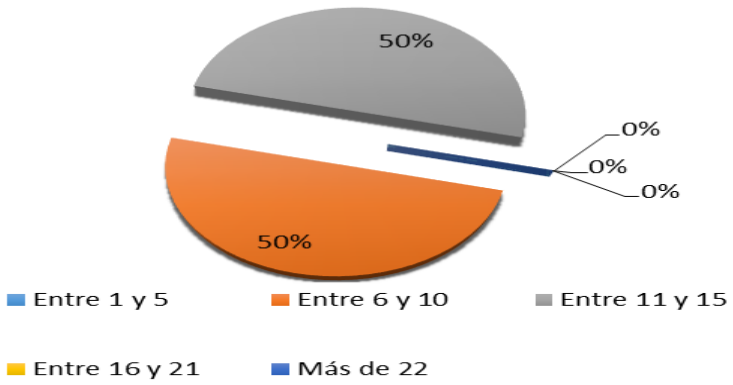


Figura 14: Representación en porciento de los resultados obtenidos tras aplicar la métrica TOC.

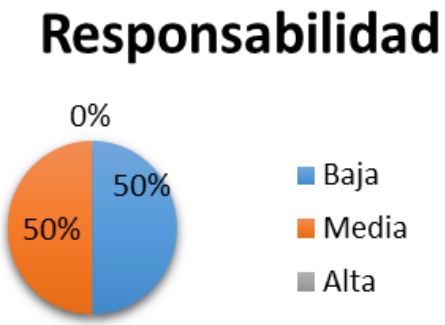


Figura 15: Valor del atributo de calidad Responsabilidad

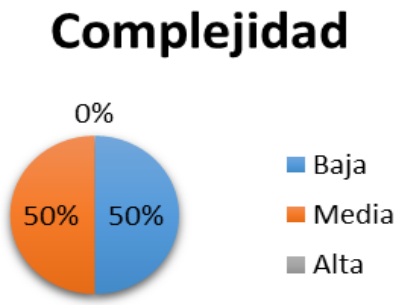


Figura 16: Valor del atributo de calidad Complejidad de implementación

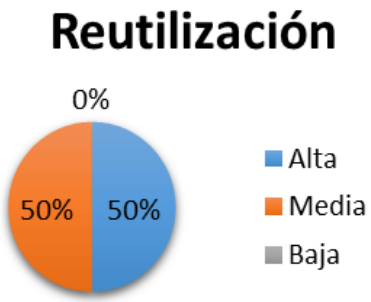


Figura 17: Valor del atributo de calidad Reutilización

Responsabilidad: luego de aplicar la métrica se obtuvieron resultados satisfactorios que reflejan una responsabilidad baja con valor del 50%.

Complejidad de Implementación: después de haberse realizado la medición de la métrica, arrojó también resultados positivos ya que la complejidad de las clases es baja en un 50%.

Reutilización: se obtuvieron valores en un nivel medio con un 50%.

Haciendo un análisis de los resultados obtenidos para los atributos de la métrica TOC se puede observar que el atributo reutilización cuenta con un porciento medio, demostrando así que el componente cuenta con

una adecuada reutilización, baja responsabilidad y complejidad en el diseño propuesto. Por lo que se concluye que los resultados obtenidos en esta métrica son positivos.

Relaciones entre Clases (RC)

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

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

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

A continuación, se muestran los resultados obtenidos de dicha métrica para una muestra de diez clases, donde al igual que la métrica anterior, solo quedaron sin analizar las clases pertenecientes a las vistas:

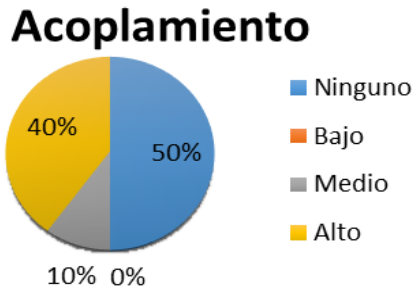


Figura 18: Grado de afectación del atributo Acoplamiento

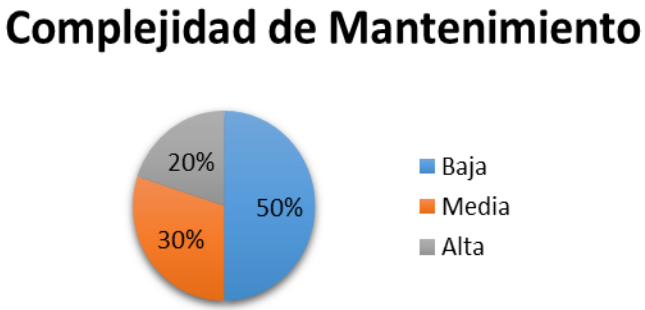


Figura 19: Grado de afectación del atributo Complejidad de Mantenimiento

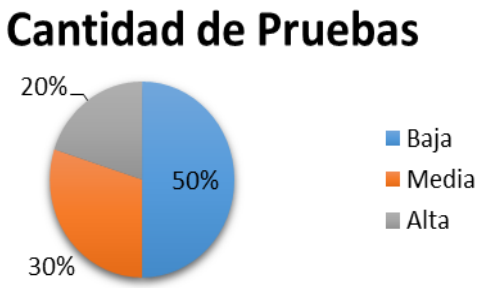


Figura 20: Grado de afectación del atributo Cantidad de Pruebas

Acoplamiento: según los resultados que se muestran, el (50%) de las clases no posee relaciones de uso por lo que no tienen valores de acoplamiento, validando una realización correcta del diseño.

Complejidad de mantenimiento: según los resultados que se muestran, el 50% de las clases se comportan de forma satisfactoria pues son de fácil soporte.

Cantidad de pruebas: luego de aplicar la métrica se obtuvo que el (50%) de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones y pruebas de software.

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

3.4 VERIFICACIÓN DEL SISTEMA

Para la verificación del sistema se realizaron pruebas de unidad con el objetivo de comprobar el correcto funcionamiento del software y a nivel de interfaz mediante el método de prueba de caja negra.

Pruebas de Liberación

Para verificar las funcionalidades del sistema propuesto se realizaron las pruebas de liberación por el método de caja negra mediante la técnica de particiones de equivalencia, a continuación, un ejemplo del Caso de Prueba correspondiente al RF registrar solicitud:

NE: Nombre del Elemento

TS: Tipo de Solicitud

TE: Tipo de Elemento

DS: Descripción de la Solicitud

RS: Respuesta del Sistema

FC: Flujo Central

Tabla 14: Caso de Pruebas Registrar Solicitud

Escenario	Descripción	TS	TE	DS	RS	FC
1.1 Registrar solicitud correctamente	Al insertar los datos requeridos se inserta la solicitud correctamente	Adicionar [Válido]	Campo [Válido]	Inicio de la asignación de implementación . [Válido]	El sistema registra la solicitud con éxito.	Página de solicitudes
1.2 Campos incompleto	Cuando se insertan datos incorrectos no se logra insertar la solicitud	[Vacío]	Campo [Válido]	Inicio de la asignación de implementación . [Válido]	El sistema muestra mensajes informando que debe ser especificado el TS.	Página de solicitudes
		Adicionar [Válido]	[Vacío]	Inicio de la asignación de implementación . [Válido]	El sistema muestra mensajes informando que debe ser especificado el TE.	Página de solicitudes
		Adicionar [Válido]	Campo [Válido]	[Vacío]	El sistema muestra mensajes informando que la DS debe ser especificada.	Página de solicitudes
1.3 Campos incorrecto	Cuando quedan datos incorrectos no	Adicionar [Válido]	Campo [Válido]	I$/	El sistema muestra mensajes	Página de solicitudes

	se logra insertar la solicitud				informando que el DS no puede contener caracteres extraños.	

Análisis de los resultados:

Para las pruebas de liberación se entregó la aplicación al grupo de calidad del centro CEGEL. Se confeccionaron 25 casos de prueba para la misma cantidad de descripciones de requisitos existentes en el sistema informático lo cual arrojó como resultado la existencia de 27 no conformidades por lo cual se hizo necesario una segunda iteración de pruebas a la aplicación, para ver el estado de las no conformidades, encontrándose que el 100% de las deficiencias encontradas habían sido resueltas. A continuación, se presentan los resultados obtenidos.

Tabla 15: Resultado de aplicar el método de caja negra

Iteración	No conformidades detectadas	No conformidades resueltas
1	27	27
2	0	0

Luego de realizar las 2 iteraciones se entregó el acta de liberación asegurando que la aplicación cumplía los requisitos mínimos para ser utilizada.

3.5 VALIDACIÓN DE VARIABLES

Como se identificó en el capítulo 1 los elementos que iban a parametrizar la variable “calidad del proceso de desarrollo de software” son las Políticas de IPP-3560:2014 pertenecientes al programa de mejora en el área de administración y configuración.

Para evaluarlas se utilizaron las listas de verificación para auditorías a la configuración. Los resultados después de aplicadas fueron que la herramienta cumple con los requisitos establecidos para el control de

cambios planteados en la lista de chequeo. Lo que demuestra que con la herramienta implementada se puede lograr un adecuado control de cambio en el proceso de implementación de las bases de datos relacionales. A continuación, en la tabla 16 se muestran los resultados:

Tabla 16: Resultado de aplicar las listas de verificación para auditorías a la configuración

Evidencia ¿Qué?	Impacto	Respuesta
¿Se encuentran los nombres y apellidos de los roles que forman parte del Comité de Control de Cambios?	Medio	Sí
¿Se describen las herramientas que se utilizan en el proyecto para controlar los cambios?	Medio	Sí
¿Se revisa si las Solicitudes de Cambio se han emitido anteriormente? ¿Se conoce qué se hace en caso que ya haya sido emitida?	Bajo	Sí
¿Existe evidencia del registro de las Solicitudes de Cambio con todos los elementos que la componen?	Medio	Sí
¿Existe evidencia de la realización del análisis de impacto del cambio solicitado y la decisión tomada?	Alta	Sí
¿Existe evidencia de la aceptación de la implementación del cambio?	Alta	Sí
¿Se comprueba la realización del cambio?	Medio	Sí
¿Se documenta claramente el cambio realizado a los productos de trabajo en el control de versiones de los documentos y en el campo descripción de la herramienta Excriba o en la descripción de la modificación en el SVN para los elementos de código?	Medio	Sí

3.6 CONCLUSIONES PARCIALES

A lo largo del capítulo se analizaron y construyeron los elementos imprescindibles que forman parte de la validación concluyendo lo siguiente:

- En los valores obtenidos en las 2 iteraciones de pruebas funcionales se comprobó que las funcionalidades diseñadas por el cliente se encontraban correctas.
- Por otra parte, al aplicar la métrica TOC se obtuvieron valores positivos para los atributos de calidad responsabilidad, complejidad y reutilización.
- La aplicación de la métrica Relaciones entre Clases mostró un bajo acoplamiento entre las clases debido a la baja dependencia que existe entre las mismas además de revelar que la complejidad para realizar mantenimiento al código de la aplicación es baja y que la cantidad de pruebas a realizar para que el diseño de las clases sea óptimo será baja.
- Por último, se validaron las variables que forman parte del problema de la investigación, demostrando que con el sistema desarrollado se cumplen los objetivos planteados.

CONCLUSIONES GENERALES

- El desarrollo del marco teórico permitió identificar los conceptos que rigen la investigación.
- El diagnóstico al proceso de desarrollo de software en CEGEL permitió identificar las deficiencias en el control de los cambios.
- Se desarrolló un sistema para el control de los cambios en el proceso de desarrollo de software de los proyectos de CEGEL, teniendo en cuenta el procedimiento de control de cambios definido en el programa de mejora de CMMI y aplicado en el centro.
- Se verificó el sistema desarrollado a partir de la ejecución de pruebas de software, obteniendo resultados positivos.

Recomendaciones

- Integrar el sistema desarrollado a aplicaciones de gestión de proyectos utilizadas en la universidad.
- Utilizar esta herramienta para el control de cambios durante el proceso de desarrollo de software de los proyectos de CEGEL con el apoyo del sistema desarrollado en este trabajo de diploma.

Referencias Bibliográficas

ABRAHAM SILBERSCHATZ, H. F. K. Y. S. S. FUNDAMENTOS DE BASES DE DATOS 2002.

ANTONIO, A. D. La Gestión de la Configuración del Software, 2001.

BABICH Software Configuration Management, 1986.

BAKKEN, S. S.; A. AULBACH*, et al.* Manual de PHP, 2001.

BAMFORD Configuration Management and ISO 9001, 1995, 1995.

Bazaar. 2009. [Disponible en: http://bzr.cvn.org.

BMC Software, 2007.

Bug Tracking and Defect Tracking Resource, 2007.

CALLEJA, M. A. A. A., MANUEL. . *Etándares de codificación*, 2009. [Disponible en: http://www.cisiad.uned.es/carmen/estilo-codificacion.pdf.

Desarrollo web. 2013. [Disponible en: http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html.

DONATO, G. M. *ClioBD. Sistema de control de versiones*

para bases de datos Universidad de las Ciencias Informáticas 2009. p.

EHEALTH, M. Remedy Change Management 2015.

FORNARIS, S., MAITE AND RABÍ, DAYANIS ALCANTARA Propuesta de una guía de métricas para evaluar el desarrollo de los Sistemas. 2009., 2009.

GONZÁLEZ, G. Sisteme de control de versiones. Caso de estudio: Subversion. Asunción: Universidad Nacional de Asunción, 2008.

Guía de los Fundamentos de la Dirección de Proyectos, 2004.

IEEE. Glosario de Terminología Informática, 2009.

JACOBSON, I. B., GRADY, RUMBAUGH, JAMES El Proceso Unificado de Desarrollo de Software, 2000.

LARMAN, C. UML y Patrones. Introducción al análisis de diseño orientado a objetos., 1999.

LINCKE, L. A. W., J. Y LÖWE Comparing software metrics tools. International Symposium on Software Testing and Analysis. , 2009.

LOZANO, V. Code Igniter En español, 2012.

MANUELE, F. A. Gestión de Cambio. Ejemplos prácticos 2012.

MEJORAS. *Programa de mejoras UCI*, 2015. [Disponible en: mejoras.prod.uci.cu

MESTRAS, J. P. Bootstrap 3.0 Aplicaciones Web/Sistemas Web, 2014.

--- Servidores Web – Apache Aplicaciones Web/Sistemas Web, 2013.

PRESSMAN, R. S. *Ingeniería del Software. Un enfoque práctico*. 2005. p.

RATIONAL Rational Software Corporation, Importancia de la Administración de la Configuración de software, 2003.

Sablime Configuration Management System Overview. 2013. [Disponible en: http://sablime.alcatel-lucent.com/

SÁNCHEZ, T. R. Metodología de desarrollo para la Actividad productiva de la UCI, 2014.

Serena Dimensions, 2014.

Serena Dimensions CM, 2016.

TECHNOLOGIES, L. The SablimeConfiguration Management System, 2006.

Bibliografía

Calisoft. 2014. Centro Nacional de Calidad del Software . Evaluación de los Productos. [En línea] 2014. [Citado el: 20 de mayo de 2014.] https://calisoft.uci.cu/index.php?option=com_k2&view=item&id=48:evaluaci%C3%B3n-de-productos.

Gaibor, Carmen. 2008. Arquitectura Multicapa. [En línea] 02 de febrero de 2008. [Citado el: 23 de enero de 2014.] http://es.scribd.com/doc/109269672/ARQUITECTURA-MULTICAPA.

Ivar Jacobson, Grady Booch y James Rumbaugh. 2000. El Proceso Unificado de Desarrollo de Software. Madrid : s.n., 2000. 464.

Laboratorio de Gestión de Proyectos. 2013. Laboratorio de Gestión de Proyectos. UCI. [En línea] 2013. [Citado el: 23 de enero de 2014.] http://gespro.cegel.prod.uci.cu/.

PostgreSQL. 1996. The PostgreSQL Global Development Group. The world´s most advanced open source database. [En línea] 1996. [Citado el: 15 de diciembre de 2013.] http://www.postgresql.org/.

Visual Paradigm. 2013. Visual Paradigm. [En línea] 16 de diciembre de 2013. [Citado el: 29 de febrero de 2014.] http://www.visual-paradigm.com/.

Granado, Yasmany García. 2012. http://repositorio_institucional.uci.cu/. http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05402_12. [En línea] 2012. [Citado el: 08 de mayo de 2014.] http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_05402_12.

RODRÍGUEZ, A. M. G. ”Propuesta de Estrategia para la Gestión de Configuración en el Proyecto Sistema de Gestión Penitenciaria”. .

Gestion de Cambios.Vision General. ttp://itil.osiatis.es/Curso_ITIL/Gestion_Servicios_TI/gestion_de_cambios/vision_general_gestion_de_cambios/vision_general_gestion_de_cambios.php.

IEEE. 2007. IEEE. [Online] 2007. **http://www.ieee.org/web/aboutus/home/index.html.**

. [ISO1995] ISO-10007. 1995. Quality Management - Guidelines For Configuration Management. 1995. 26. ISO-12207.

ISO 12207.ISO-9000. Selección y uso de la tercera edición de las normas ISO 9000.kanav. http://www.kanav.com. tp://www.kanav.com. [Online] http://www.kanav.com.

SWEBOK. Guide to the Software Engineering Body of Knowledge. California, EEUU, 2004.

JACOBSON, I. El Proceso Unificado de Desarrollo Software, 2000. [Disponible en: http://bibliodoc.uci.cu/pdf/reg03050.pdf

CASE Corporativo para la creación de la línea base de una empresa de Software. Revista Ingeniería Industral, CUJAE, 2000b.

MConfig.PM, Modelo de referencia para la Gestión de Configuración en la pequeña y mediana empresa de software. La Habana, Cuba, CUJAE, 2004. p.

GERVÁS, P. Estándares y Gestión de Configuración. UCM, 2002. p.

ANEXOS

Entrevista:

Nombre del centro: CEGEL

Preguntas:

Existe un administrador de la configuración

1.1- ¿Cuántas bases de datos existen en el proyecto?

1.2- ¿Se usa una base de datos centralizada o cada desarrollador tiene una copia local de la base de datos?

¿Cuántas personas pueden cambiar la estructura de la base de datos?

2.1- ¿Qué roles ocupan estas personas?

¿Los cambios aplicados se producen arbitrariamente o pasan por un proceso de aprobación?

3.1-En caso de pasar por un proceso de aprobación responda:

3.1.1-¿Quiénes aprueban dicho proceso?

3.1.2-Explique cómo se ejecuta este proceso de aprobación

¿Se notifica al resto del equipo los cambios realizados?

4.1-¿Qué medios se utilizan para realizar la notificación?

¿Existe un mecanismo para llevar una trazabilidad de los de los cambios en la estructura de las bases de datos?

¿El proceso de salva o Restauración de la base de datos se realizan de forma manual o se encuentra automatizado?

6.1- Explique cómo se realiza este proceso.

¿Cree necesario realizar el control de cambios en el desarrollo de las bases de datos?

¿Mencione los elementos que a su consideración no deberían faltar para el control de cambio en la estructura de las bases de datos?

Tabla 17: Entrevista a diversos trabajadores de CEGEL

	¿Existe un Administrador de la Configuración?	Cantidad de BD. ¿Quiénes cambian la estructura de la BD?	¿Los cambios pasan por un proceso de aprobación? ¿Quiénes lo aprueban?	¿Se notifican los cambios? ¿Por qué medios?	¿Existe un mecanismo de trazabilidad de cambios?	¿La BD es centralizada?	¿El proceso de salva es manual?
Jefe de proyecto de Fiscalía	Sí	Cantidad – 1 El arquitecto de datos o quien ocupe su lugar.	Sí Arquitecto de datos, de software y jefe de desarrollo. Casos drásticos el analista y jefe fe proyecto	Sí Verbal	No	Sí	Internamente se realizaban salvas manuales

Jefe de proyecto de Tribunales	Sí	Cantidad – 1 Administrador de BD o a quien este de permisos	Sí Jefe de proyecto, arquitecto de software, administrador de BD, jefe del módulo afectado	Sí Verbal	No	Sí	Automáticas
Jefe de proyecto de Fiscalía. (Actualmente)	No	Cantidad – 1 Los 2 desarrolladores y los 2 administradores	Sí Arquitecto de datos y los desarrolladores	Sí Verbal	No	Sí	Manuales
Administrador de BD de Decisiones de Fiscalía	No	Cantidad – 1 Solo el Administrador de BD	Sí Jefe de proyecto, analista principal y administrador de base de datos	Sí Verbal	No	Sí	Por necesidad se han hecho manuales pero existe una tarea automatizada

Arquitecto de datos de Fiscalía	No	Cantidad – 1 Solo el Administrador de BD	Sí Analista, desarrollador principal y arquitecto de datos	Sí Verbal	No	Sí	Manuales
Jefe de proyecto de Tribunales (Actualmente)	No	Cantidad – 1 El arquitecto de datos	Sí Analista, desarrollador principal, arquitecto de datos, y el de datos correspondiente al módulo	Sí Verbal	No	Sí	Automáticas

Aplicación de las listas de chequeo

Leyenda:

FS = Fecha de solicitud

RS = Requisito solicitado

Spor = Solicitado por

R = Resultado

A = Aprobado

PRV = ¿El proveedor del REQ es un proveedor válido?

RIU = ¿El REQ está identificado como único?

RM = ¿El REQ es modificable?

RNA = ¿El REQ no es ambiguo?

RC = ¿El REQ está completo?

RCong = ¿El REQ es congruente con otros REQ relacionados?

RI = ¿El REQ puede ser implementado?

RP = ¿El REQ puede ser probado?

EvaP = ¿El resultado de la evaluación de impacto es positivo?

RCorr = ¿El REQ está correcto?

RT = ¿El REQ es traceable?

C = Cliente

Tabla 18: Aplicación de la lista de chequeo

RS	Spor	FS	PRV	RIU	RM	RNA	RC	RCong	RI	RP	EvaP	RCorr	RT	R
RF1	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF2	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF3	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF4	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF5	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF6	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF7	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF8	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF9	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF10	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF11	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF12	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF13	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF14	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF15	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF16	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF17	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF18	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF19	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF20	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF21	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF22	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF23	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF24	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A
RF25	C	15/04/2016	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	A