Master universitario en investigación en ingeniería de software y sistemas Informáticos.

ETS de ingeniería informática

El gobierno del software en fase de mantenimiento

Erwin Daniel Suarez Rodriguez

Tutor:

María Magdalena Arcilla Cobián

Madrid, septiembre de 2017

Master universitario en investigación en ingeniería de software y sistemas Informáticos.

ETS de ingeniería informática

Itinerario Ingeniería de Software (Código 31105128)

El gobierno del software en fase de mantenimiento

Trabajo Fin de Master tipo B

Erwin Daniel Suarez Rodriguez

Tutor:

María Magdalena Arcilla Cobián

Madrid, septiembre de 2017

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 30/06/2017

Quién suscribe:

Autor(a): Erwin Daniel Suarez Rodríguez
D.N.I/N.I.E/Pasaporte.: 88.224.520 (Expedido en Colombia)

Hace constar que es la autor(a) del trabajo:

El gobierno del software en fase de mantenimiento.

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

IMPRESO TFDM05_AUTOR
AUTORIZACIÓN DE PUBLICACIÓN
CON FINES ACADÉMICOS

Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Agradecimientos

Cada grado académico me lleva a cuando todo comenzó. A la profesora Oliva. Quien tenía el mismo nombre de mi madre y fue la primera que me soportó en un salón de clases. Años ochenta, Táriba: un pueblito encaramado en la cordillera andina en el occidente de Venezuela. Me enseñó cosas muy importantes que debe saber cualquier persona: entender lo que leía, sumar, comportarme, y a vivir de forma responsable la libertad.

Desde entonces, han sido muchos mis maestros y perdí la cuenta de los libros que me han enseñado. La docencia, para mí, es un apostolado. Personas que eligieron enseñar como forma de vida y de su labor diaria han salido, salen, y saldrán; artesanos, costureras, ingenieros, médicos, abogados, padres… Personas.

Así que este trabajo lo agradezco a aquellos que a diario se dedican a tratar que aprendamos cosas útiles.

¡Gracias profe!

Y

Muchas gracias profesora Oliva.

Resumen

La fase de mantenimiento del software consiste en mantener en operación los programas y construir, probar e instalar cambios al mismo. Es una etapa que puede durar por toda la vida de la organización que usa el software. Gobernar con eficiencia el mantenimiento es un gran problema para la mayoría de directivos de tecnologías de la información. Se propone la gestión con un cuadro de mando integral. Concepto de las ciencias administrativas que parte de la estrategia de la organización y la convierte en indicadores que muestran el cumplimiento de estos objetivos. Este tipo de gestión ha mostrado mucho éxito en organizaciones de todo tipo. Para la implementación se propone la organización de unas áreas con marcos probados de diferentes metodologías y marcos de proceso. El cuadro de mando integral es similar al tablero de instrumentos de una nave. Con él, la tripulación puede tomar decisiones inmediatas para corregir el rumbo que lleva al destino: la estrategia organizacional. Este sistema de gobierno es estratégico: busca optimizar el uso de recursos siempre escasos. Reduce la paradoja de la productividad que afirma que no se ven los resultados de las inversiones en tecnología, guiando la gestión hacia la consecución de resultados.

Palabras Clave: cuadro de mando integral, gestión de tecnologías de información, procesos de software, desarrollo ágil, administración de la configuración, mantenimiento del software, paradoja de la productividad.

Abstract

The maintenance phase of the software consists of keeping the programs in operation and constructing, testing and installing changes to it. It is a stage that can last for the life of the organization that uses the software. Effective maintenance management is a major problem for most information technology managers. Strategic management is proposed with a balanced scorecard. Concept of the administrative sciences that starts from the strategy of the organization and turns it into indicators that show the fulfillment of these objectives. This type of management has shown great success in organizations of all kinds. For the implementation, it is proposed the organization of process areas with proven frameworks of different methodologies and process frameworks. The balanced scorecard is similar to a dashboard on a ship. With it, the crew can make immediate decisions to correct the direction that leads to the destination: the organizational strategy. This system of government is strategic: it seeks to optimize the use of scarce resources. It reduces the productivity paradox, that says that the results of the investments in technology are not seen, guiding the management towards the results.

Keywords: Balanced scored card, IT management, Software processes, Agile development, Configuration management, Maintenance of the software, productivity paradox.

Índice

Lista de figuras

1 Introducción

El software está presente en casi todos los aspectos de nuestra vida. Bancos, hospitales, escuelas, organizaciones estatales… Tienen uno o varios programas de computadora que apoyan los procesos de la organización.

Este software cambia. Bien sea por regulaciones estatales, cambios de tecnología o nuevas maneras de hacer las mismas cosas el software debe adaptarse. Ese proceso de cambios se llama mantenimiento y es la fase más larga del ciclo de vida. Puede incluso extenderse por toda la vida de la organización.

Si a esto sumamos que, con la naturaleza global de los negocios de tecnologías de información, los clientes son mucho más exigentes en cuanto a tiempo de entrega y calidad del producto. Estamos ante un panorama muy difícil para quienes se dedican al mantenimiento de los programas.

Mucho del software que tienen las organizaciones en producción, son de construcción de hace más de cinco años, y la mayoría se fabricó sin un enfoque de procesos o una metodología formal. En general se tiene un conjunto de código fuente, ejecutables e infraestructura y sobre esto se deben hacer los cambios.

El equipo encargado de hacer los cambios bien sea dentro de la misma organización, o un equipo externo, tiene en la misma bolsa de solicitudes: reportes de errores, solicitudes nuevas y déjelo como estaba. Administrar esto es un caos para la mayoría de gerentes de tecnologías de información (en adelante TI), y el usuario del software se siente defraudado con un equipo muy costoso.

El presente estudio plantea lo que se debe tener en cuenta para lograr la operación adecuada de un software productivo, cómo hacer los cambios de forma ordenada, predecible y repetible. Y gestionar la operación del mismo optimizando los recursos, siempre limitados con que se cuente.

Toma conceptos de ciencias administrativas: modelos de negocio, estrategia empresarial y cuadro de mando integral. De patrones de diseño: para la administración de la configuración del software. Del enfoque ágil: para administrar las solicitudes de cambio. De marcos de proceso: ITIL, CMMI, ISO Spice y PMP para las liberaciones de producto, administración del soporte y los activos.

Todo esto se plantea sea gobernado por un cuadro de mando integral, que es un conjunto de indicadores que dan una visión general del estado de TI y permite tomar decisiones en tiempo real y alinear los desfases con los planes.

1.1 Objetivos

El mantenimiento del software no puede ser tratado como un proyecto dado que no tiene un final fijo en el tiempo. Es una fase que puede durar toda la vida de la organización. Por lo que el presente estudio propone:

1. Mostar la forma en que las organizaciones realizan el mantenimiento del software

2. Proponer el establecimiento de áreas de proceso para la operación y los cambios del software.

3. Proponer un cuadro de mando para gobernar el mantenimiento del software.

El Cuadro de Mando Integral o CMI es un concepto de las ciencias administrativas. Consiste en una serie de indicadores, no más de veinte, que dan una visión general del estado de la empresa. El presente estudio propone un cuadro de mando para el gobierno de TI.

Los indicadores del CMI se alinean con los objetivos de la organización de modo que se busca la optimización de recursos

1.2 Estructura

El trabajo de fin de máster se divide en seis capítulos: Introducción, Planteamiento del problema, Estado de la cuestión, Resolución, Conclusiones y Líneas de investigación futuras.

En el planteamiento del problema se describe la problemática que se tiene con la fase de mantenimiento, que los modelos de proceso no dan una respuesta satisfactoria y completa a los cambios y la operación diaria en su conjunto.

El estado de la cuestión describe la forma como las organizaciones han enfocado la fase de mantenimiento. Con modelos de procesos, en modo resolver el día a día e implementar una herramienta de desarrollo siguiendo las instrucciones del fabricante de la herramienta.

La resolución se divide en nueve subcapítulos, que cubren los temas que debe tener en cuenta un grupo que realiza cambios en un software productivo: El negocio y la estrategia, administración de la configuración, desarrollo ágil, aseguramiento de la calidad, gestión de cambios del software, gestión de activos, soporte y manejo de errores, gestión del conocimiento y el cuadro de mando integral.

Cada uno de estos subcapítulos de la resolución, se basa en conocimiento existente y temas tratados en el desarrollo del máster. La administración de la configuración se trata con patrones de diseño, tema de amplio conocimiento en arquitectura del software. El enfoque ágil, la gestión de la calidad se tratan en Mejora de procesos de software y desarrollo de software seguro. También se incluyen conceptos tratados en especificación. Al principio de cada subcapítulo hay una figura que resume los pasos para implementar la solución propuesta.

Las conclusiones muestran lo que se encontró en el desarrollo del estudio.

En las líneas de investigación futuras se plantean dos temas: El costo de TI y el valor del conocimiento del personal de TI. Ambos requieren de desarrollo investigativo. Se han realizado grandes inversiones en el área de TI de muchas organizaciones sin tener la certeza si estas inversiones están produciendo un retorno. Por otra parte, la flexibilización laboral ha hecho que haya una alta rotación de personal en el área lo que hace crítica la perdida conocimiento por parte de las organizaciones. Dadas estas circunstancias se plantea también el tema de la ética profesional de los trabajadores de TI. ¿Cómo se controla? ¿Qué entidades hacen algo al respecto?

2 Planteamiento del problema

2.1 Las aplicaciones que están hoy en producción

Tenemos una enorme cantidad de software en producción y que fue construido durante las últimas cuatro décadas en Hispanoamérica.

Estas aplicaciones son vitales para las organizaciones que las usan. Bancos, hospitales, escuelas, instituciones del gobierno y todo tipo de empresas, tienen una alta dependencia en su operación de estos programas. La mayoría de este software fue construido empleando el conocimiento que se tenía entonces y muy poco se fabricó con una metodología formal o de procesos.

Así se tiene en general: un código fuente, unos ejecutables, y una infraestructura que sirve ese software (servidores, redes, software base).

Cambiar este software por uno más actualizado, en la mayoría de los casos no es una opción, ya que está adaptado al negocio y ese cambio tendría un alto costo de tiempo y dinero.

Pero el software cambia. Normas, procesos y tecnologías hacen que el software deba adaptarse. Y al hacer los cambios el software debe continuar en operación, ya que es vital para la organización que lo usa.

2.2 La forma tradicional de hacer los cambios del software productivo y de vieja data

En general, las organizaciones contratan a un grupo de desarrolladores, bien sea dentro de la organización o a un externo, y les entregan “lo que hay”. Que suele ser: código fuente, algo de documentación y alguna infraestructura física. A esta fase de cambios se le llama mantenimiento. Y el departamento de TI o la fábrica de software tiene en una misma bolsa: nuevas solicitudes, errores, “déjelo cómo estaba” y la administración del día a día.

Gobernar esto se vuelve un caos. Y la respuesta general de los gerentes de TI es vivir en modo “apaga incendios”. Desarrolladores revisan errores de producción, la gente de pruebas explica por qué el despliegue de la nueva versión falló. Se ajusta el cronograma que nunca se cumple. El administrador de la red trata de encontrar la explicación del alto tráfico. Se realizan reuniones interminables sin ninguna conclusión. Y se acumulan más y más solicitudes. Al final se tiene la percepción de un equipo de TI muy caro e inútil.

2.3 Los modelos de procesos no dan una respuesta completa

Los modelos de proceso traen grandes ventajas para las organizaciones que las implementan. Facilitan la planificación, el control de los proyectos, optimizan el uso de recursos y facilitan la evaluación. Son muy buenos para la construcción de software desde cero, pero no para una labor diaria de operación del mismo. Y hay que considerar que el software cambiará.

Los estándares de modelos de procesos existentes y más populares: ISO Spice, CMMI, PMP, no dan una respuesta completa a esta dificultad. Estos modelos tratan de establecer practicas a grupos que desarrollan software nuevo. Que no está en producción. Y tratan la fase de mantenimiento de manera superficial. Algunas organizaciones han aplicado los mismos procesos iniciales para su fase de mantenimiento, encontrándose con procesos tediosos, burocráticos y lentos para resolver situaciones del día a día.

Por ello han surgido nuevas herramientas para resolverlo: administración de la configuración e integración continua son algunas de las repuestas a algunos de los problemas de gobierno de TI. Pero aun así son respuestas sueltas sin considerar que las organizaciones son un todo.

3 Estado de la cuestión

El software está en todos lados. E influye en nuestra vida. Bancos, escuelas, instituciones de salud, y sea cual sea la organización para la que trabajemos o necesitemos un producto o servicio. Hay un tipo de software que soporta/apoya el negocio. Y si el desarrollo esperado del *Internet de las cosas* se da, tendremos software en todo lo que nos rodea.

Todo este software tiene algo en común. Uno, fue hecho por personas. Dos, cambia con el tiempo. Tres, necesita recursos humanos y tecnológicos para operar.

El proceso de cambios de un software productivo y la administración de su operación es un dolor de cabeza para las organizaciones que lo hacen. Para afrontarlo, en general han tomado varios caminos:

1. Vivir el día a día en modo apaga incendios.
2. Implementar algún modelo de procesos.
3. Implementar una herramienta de desarrollo y seguir las instrucciones del fabricante.

3.1 Vivir el día a día en modo apaga-incendios

Es el enfoque que tiene más problemas, y en el que viven muchos equipos de TI. Esperan que aparezca un problema, analizan y dan una solución. Esta solución pasa desde un aumento de la capacidad de hardware hasta una modificación en el programa.

Este enfoque puede producir más problemas que lo que resuelve. Nuevos errores causados por la solución, viejos errores reaparecen y acumulación de solicitudes suelen ser lo más común cuando se aplica esta forma de trabajar.

3.2 Implementar algún modelo de procesos

Otra forma de afrontarlo es implementando en el área de TI un modelo de procesos.

3.2.1 Los modelos de madurez

CMMI [1] e ISO SPICE [2]. Son modelos de madurez. Es decir, que según la organización implemente procesos tiene un nivel de madurez que va de 1 a 5 para CMMI y del 1 al 3 para ISO SPICE. Son populares en la industria y en varios países de Hispanoamérica han tenido apoyo del estado para su implementación.

Ninguno de estos dos enfoques toca en detalle el tema de mantenimiento del software. Ninguno de sus procesos o prácticas habla en concreto de cómo mantener el software. Y si a esto se suma que dicen el que, pero no el cómo de los procesos, tenemos un cuello de botella a la hora de hacer un cambio a las aplicaciones.

Las organizaciones que tienen un nivel de madurez de uno de estos modelos. Han tomado la opción de realizar el mantenimiento, con los mismos procesos que para hacer el software nuevo; es decir, cada cambio pasa por todo el flujo de trabajo. Haciendo que los cambios sean muy lentos y no se ajusten a la realidad de los negocios

3.2.2 ITIL

ITIL [3] es un conjunto de buenas prácticas para la gestión de servicios de tecnologías de la información.

Está dirigido a la administración de la infraestructura tecnológica, definiendo un conjunto de servicios que deben ser soportados por el equipo. Cada uno de estos servicios tiene procesos.

El proceso de gestión del cambio, parte del servicio transición del servicio, es lo que está más cerca del mantenimiento del software. Sin embargo, cubre cómo hacer el cambio afectando lo mínimo posible la operación, pero no, como construir ese cambio.

3.2.3 COBIT

COBIT [4] Objetivos de control para la información y tecnologías relacionadas. Es un conjunto de prácticas para el gobierno de TI. Su implantación ayuda en la gestión de recursos.

Los procesos de los dominios de Adquisición e implantación y Entrega y soporte cubren parcialmente el mantenimiento del software. Pero COBIT se limita a gobernar a través de indicadores, políticas y auditorías. Tampoco da el cómo hacer los cambios.

3.2.4 PMP

PMP [5], Project Management Professional. Es una formación y certificación en buenas prácticas para gestores de proyectos de todas las disciplinas. Es promovido por el PMI, Project Management Institute. Y consiste en una serie de procesos que un gestor de proyectos debe implementar para dirigir proyectos con éxito.

La dificultad radica en que un proyecto tiene un fin en el tiempo. Y enfocar el mantenimiento del software como un proyecto da mucho trabajo burocrático inútil. Por ejemplo, no es posible definir el alcance si pueden introducirse cambios en cualquier momento.

Sin embargo, muchas empresas han tomado este enfoque y tienen una oficina de proyectos que se encarga de dirigir el proceso de cambios. Al igual que con los modelos de procesos anteriores hay cambios muy lentos que no se ajustan a la necesidad del negocio.

3.2.5 Implementar una herramienta de desarrollo y seguir las instrucciones del fabricante

Los grandes fabricantes de lenguajes de programación, Oracle, Microsoft, Borland, IBM. Y los proveedores de servicios en la nube: Oracle, Google, Amazon, Microsoft. Ofrecen herramientas para administrar el ciclo de vida del software.

Estas herramientas cubren desde el entorno de desarrollo hasta la puesta en producción del software, pasando por las etapas de análisis, desarrollo y verificación del software.

Ofrecen herramientas automatizadas y otras manuales para la gestión.

Aunque son muy buenas y la mayoría han tomado un enfoque ágil de desarrollo tienen las siguientes desventajas:

1. Son costosas.
2. Se limitan a una sola tecnología y lenguaje de programación.
3. Imponen procesos que no necesariamente se alinean con el negocio.

Estas desventajas hacen que, aunque sean mucho más ágiles que los enfoques anteriores. Aun ofrezcan dificultades a la organización por no tener flexibilidad a la hora de cambiar de tecnología o tener una reducción en el presupuesto.

Teniendo esto en cuenta se plantea una solución que modela una serie de áreas que debe organizar un grupo de TI y que se gobiernan con un cuadro de mando integral.

4 Resolución

Gobernar la fase de mantenimiento es una actividad que consume al personal de TI. Varias versiones del producto, errores conocidos y desconocidos, solicitudes de los clientes/usuarios, cambios de normas, fallos de seguridad…Y mantener el software operando. Hacen que el día a día de muchas organizaciones sea un caos.

En la presente memoria se proponen una serie de áreas que facilitarán esta administración haciéndola medible y reproducible. Se basan en diferentes marcos de procesos que han mostrado éxito a través del tiempo, y que en general las empresas tienen en alguna medida implementados.

Se propone que estas áreas sean gestionadas a través de un cuadro de mando integral, herramienta que alinea la estrategia organizacional con las actividades diarias de la organización.

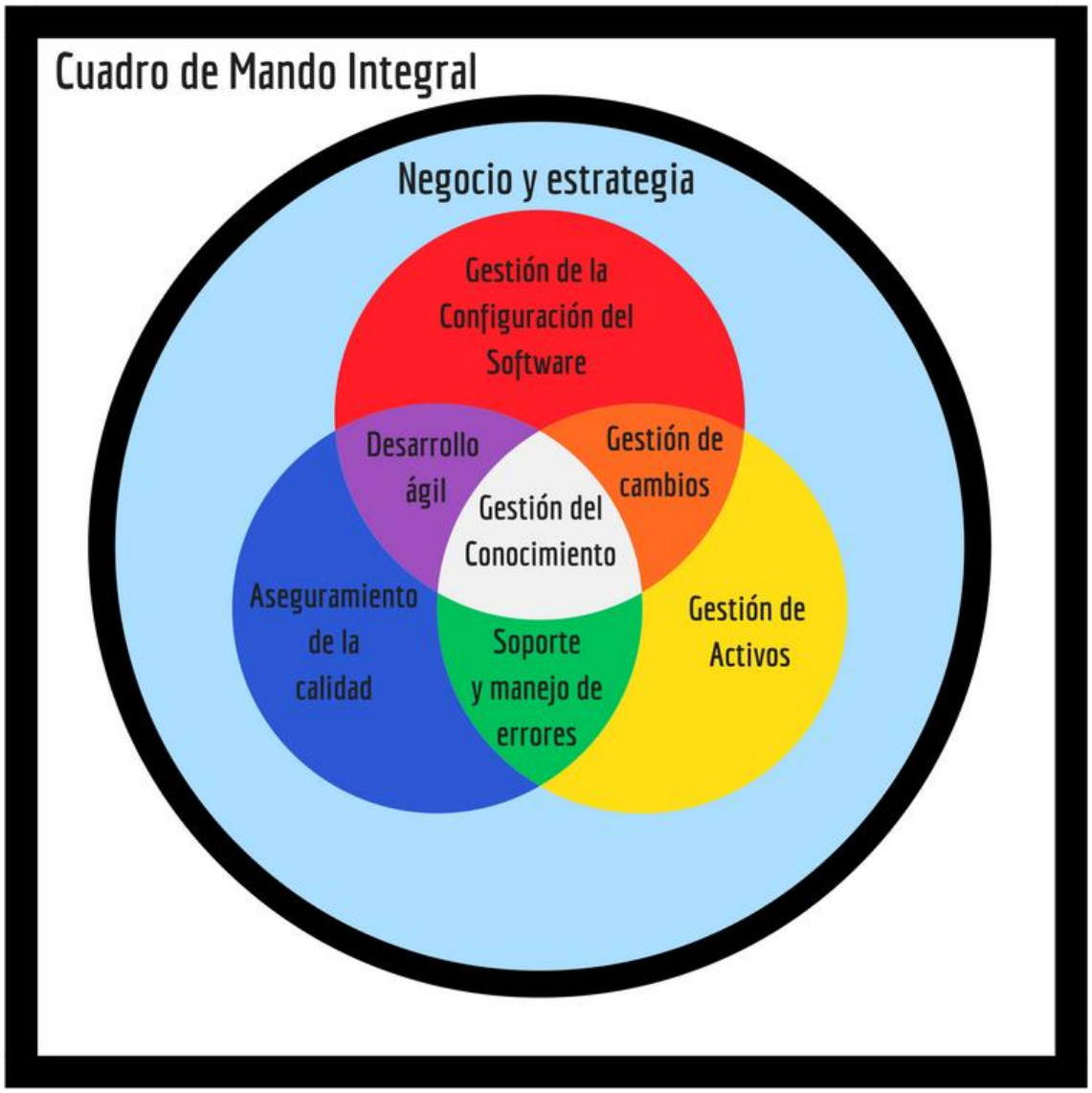

Figura 1. Gobierno del software en mantenimiento.
[Elaboración propia]

La figura Gobierno del software en mantenimiento grafica la propuesta. Pone en el centro la Gestión del conocimiento, ya que el saber, acerca de cómo modificar, gestionar, configurar, errores conocidos, debe estar al alcance de todos los procesos.

Alrededor de la gestión de conocimiento se plantean seis actividades:

1. Administración de la configuración del software. encaminada a la organización de las herramientas necesarias para la construcción/modificación del software.
2. Desarrollo ágil. Marco de desarrollo que busca liberaciones frecuentes y útiles del software.
3. Aseguramiento de la calidad. Encargado de la aplicación de pruebas y de la medición y evaluación de métricas.
4. Gestión de cambios del software. Para controlar la puesta en producción de los cambios realizados con el mínimo impacto a la operación.
5. Gestión de activos. Para administrar los activos tangibles e intangibles necesarios para la operación.
6. Soporte y administración de errores. Para conectar los usuarios y el personal de TI en el uso del software. Y dar manejo a los errores encontrados.

Estas áreas deben estar orientados a la estrategia del negocio y gestionadas en un cuadro de mando integral, que son un conjunto de métricas e indicadores que muestran el cumplimiento de la estrategia.

No se debe confundir el cuadro de mando integral con los indicadores de proceso. El CMI es una herramienta de gestión, que la administración usa para reorientar el trabajo si hay desviaciones, o mostrar, que en efecto se está alcanzando la meta. No se debe caer en la tentación de castigar individualmente al personal por un mal desempeño de las métricas del CMI.

Por otra parte, los indicadores de proceso muestran la efectividad de los procesos, y deben ser usados para mejorar la forma en que se hacen las cosas. La implementación de un CMI no significa la eliminación de los indicadores de proceso.

Y si la organización tiene un plan de medición del desempeño individual, se tendrán indicadores de objetivos personales.

La meta de un CMI es que el día a día de la organización se oriente al cumplimiento de la estrategia optimizando los recursos disponibles.

En la *figura Pasos para establecer gobierno del software en mantenimiento*, se muestran los pasos para implementar este gobierno del software operativo y en mantenimiento. Y a continuación una muestra de un cuadro de mando.

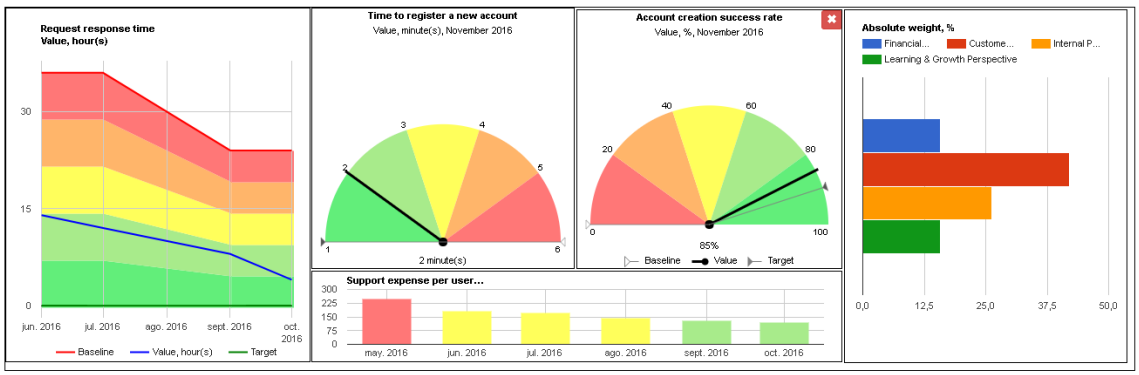

Figura 2. Muestra de indicadores de cuadro de mando integral
[Fuente: https://www.webbsc.com]

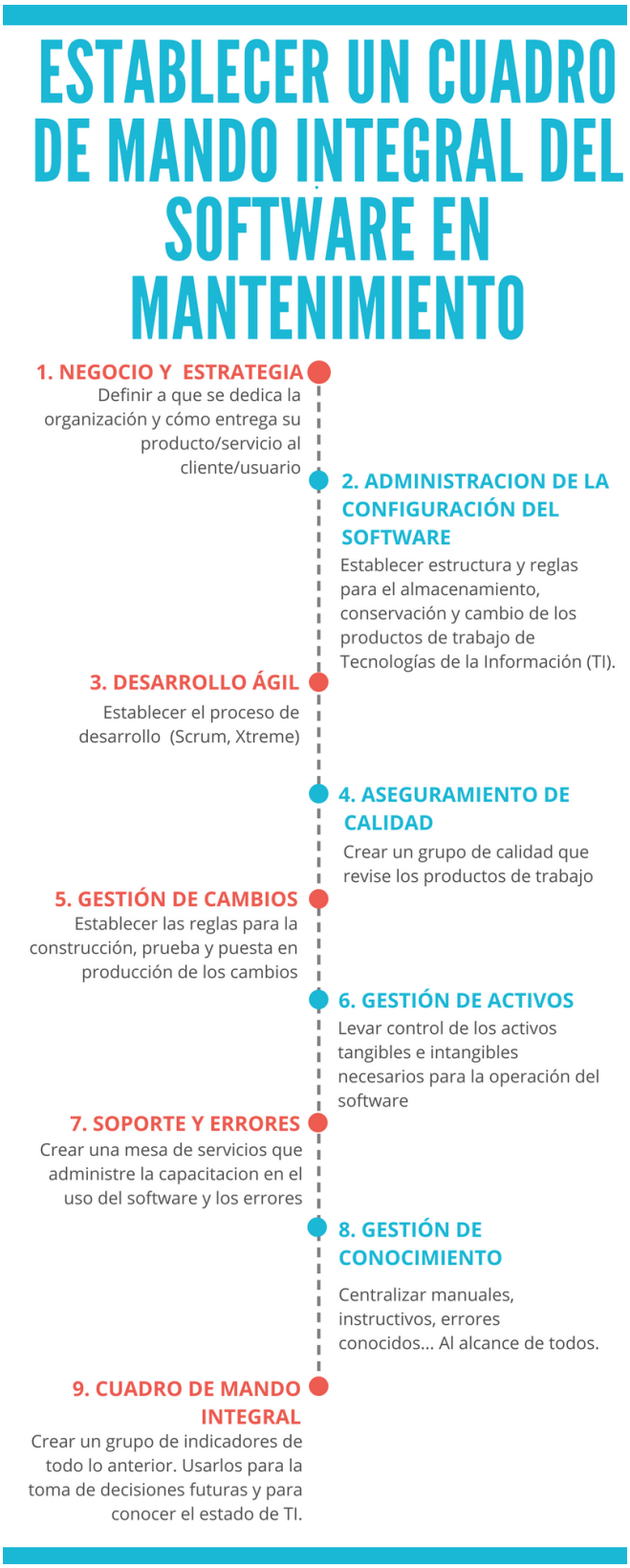

Figura 3. Pasos para establecer un cuadro de mando integral del software en mantenimiento [Elaboración propia]

En los siguientes capítulos se detalla acerca de cada uno de estos pasos, mostrando metodologías probadas para la implementación.

Al final de cada uno de los procesos propuestos se muestran ejemplos de métricas que pueden ser usadas para cada proceso.

4.1 EL NEGOCIO Y LA ESTRATEGIA

Figura 4. Pasos para determinar el negocio y la estrategia [Elaboración propia]

Lo primero que debe establecerse pa elaborar un cuadro de mando es establecer el negocio y la estrategia del negocio, y de que manera el software ayuda al negocio e impulsa la estrategia.

El software consiste, en un conjunto de programas de computadora que se ejecutan en un grupo de máquinas de cómputo y hacen algo útil. Ese algo útil suele ser la

automatización de un proceso industrial/administrativo, o la ejecución de entretenimiento.

Los programas usados por las empresas ayudan a hacer más eficiente el negocio al que se dedican. Bien sea este un producto o un servicio, el software automatiza las tareas repetitivas y predecibles del negocio, dando como resultado que la empresa sea más productiva. Al menos esto es lo que debería hacer.

La dinámica empresarial hace que las formas de hacer las cosas cambien. Por lo que el software debe adaptarse a esas nuevas necesidades. En las organizaciones suele manejarse el software con un enfoque de proyecto. Lo que es erróneo ya que una vez implementado se convierte en la operación diaria de la empresa.

Es más eficiente gobernar TI desde el punto de vista de operaciones que de proyectos. Los proyectos tienen un límite en el tiempo, las operaciones duran mientras la organización exista.

Lo primero que debe hacerse es alinear TI con el negocio, solo así será eficaz su operación.

Como consecuencia de lo anterior surgen dos axiomas acerca del software:

1. **Todo software hace algo útil.**

Bien sea la impresión de una carta, la ejecución de un juego o el control de un dispositivo médico. El software hace algo para alguien.

2. **El fin de la tecnología es hacernos más productivos.**

Por ejemplo, un contador que trabaja a mano lleva la contabilidad de diez empresas. Usar un software debe permitirle multiplicar ese trabajo. Si el software que usa no le permite hacer diez veces más, no es aceptable. Cien empresas es lo aceptable para justificar la inversión en tecnología.

La paradoja de la productividad

Sin embargo; esto no es lo que sucede en el mundo real. La productividad no ha aumentado de forma considerable a pesar de las enormes inversiones en tecnología. Este problema fue denominado la *Paradoja de la Productividad o Paradoja de los Ordenadores de Solow.*

Robert Solow. Economista estadounidense planteó que se puede ver el establecimiento de los ordenadores en todos los sitios menos en los gráficos de productividad [6]

Desde entonces varios autores han estudiado el fenómeno, haciendo diferentes consideraciones, siendo Brijolfsson [7] el más citado año tras año. [8]

Brijolfsson plantea cuatro posibles explicaciones a la paradoja de la productividad. Uno, que hay retrasos en los retornos de la inversión, que este retorno solo es posible en el largo plazo. Dos, errores en la medición, no se están tomando en cuenta todas las variables para medir el retorno de la inversión. La tercera explicación plantea que las inversiones en tecnología y su retorno se anulan mutuamente y, en cuarto lugar, la explicación de la que se quiere encargar la presente investigación: Errores en la gestión. [8]

Las investigaciones acerca de la paradoja de la productividad no son concluyentes, el reto para los investigadores es encontrar datos que respalden o refuten la paradoja y sus explicaciones. Pero cierta o no es la sensación existente en la mayoría de organizaciones, sin importar si estas son públicas o privadas.

4.1.1 ¿Qué es el negocio?

En el presente trabajo se llama negocio a la razón principal de la existencia de la organización. Sin importar si este es comercializable o no. En esta definición caben las empresas de productos y servicios, que esperan una retribución económica. Y las instituciones estatales, organizaciones no gubernamentales y todas las organizaciones humanas que existen para dar soporte a algún punto de la civilización.

Desde ese punto de vista:

El negocio es lo que las personas esperan de la organización, bien sea un producto o un servicio. Incluso están dispuestas a pagar por obtener ese algo.

El software debe hacer más productivo el negocio. Lo debe facilitar, automatizar las tareas repetitivas y generar información fiable. Este concepto está muy relacionado con el de modelo de negocio, que se tratará más adelante.

Aunque parece una obviedad, hay organizaciones que no tienen claro de que se trata su negocio, y se pierden en procesos complejos, formatos inútiles y reuniones interminables.

Ejemplos de negocios son:

- Las aerolíneas transportan personas de un aeropuerto a otro.
- Los intermediarios inmobiliarios ayudan a las personas a encontrar donde vivir.
- La oficina de pasaportes expide, renueva y modifica un documento de identificación internacional para los ciudadanos.
- Las universidades certifican la idoneidad profesional de sus egresados.

Una organización puede tener más de un negocio, pero en general hay uno principal y los demás son derivados de este, por ejemplo; las universidades, adicional de certificar a sus egresados realizan investigaciones. Lo importante aquí es identificar el negocio principal en una frase o dos, e identificar los negocios secundarios.

Todas las organizaciones están manejadas por personas, entre más grandes, más personas y más profesiones se mezclan para hacer posible la organización. Gerentes, directivos, PMO, consejeros delegados, financieros, contadores, asesores comerciales y en alguna parte el personal de TI.

Entre metodologías de calidad, procesos y manuales de funciones cada uno se dedica a lo que sabe. El comercial consigue clientes, el financiero negocia con los bancos, los contadores estructuran impuestos, y todos. Les hacen solicitudes a tecnologías de la información. En esta complejidad es posible que en un momento se olvide cual es el negocio y se dediquen a ejecutar procesos.

4.1.2 Modelar el negocio

La palabra negocio tiene una connotación deshumanizante “no es nada personal, solo son negocios” es una frase común. En este trabajo se pretende ampliar el concepto para definirlo cómo la razón de ser de la organización.

El modelo de negocio es la representación abstracta de la organización, donde confluyen aspectos financieros, productos, segmento de clientes y todo lo que lleve a alcanzar las metas y objetivos organizacionales. En este concepto caben las empresas que esperan obtener beneficios monetarios, y también las que sus beneficios son de tipo no monetario, cómo organizaciones humanitarias, instituciones estatales, educativas e incluso militares. Todas, tienen objetivos que cumplir. Y estos objetivos hacen realidad el modelo de negocio.

Sin un modelo de negocio es improbable que la organización sobreviva. Si no lo tiene corre el peligro de desaparecer, bien sea porque su rentabilidad no existe, o porque el superior jerárquico decide liquidarla. Una organización que no tiene claro para que existe desaparecerá.

El modelo de negocio es la forma como una organización da valor a sus consumidores. Puede estar definida en una frase, en todo un manual de negocio, o en una serie de procesos. E incluye los productos o servicios prestados, los aspectos financieros y el público al que está dirigido.

Con base a este se realiza el modelado de negocio, también conocido como cadena de valor.

La cadena de valor

Modelar el negocio es un concepto del lenguaje unificado de modelado: UML.

En él, a través de diferentes diagramas se describe la forma cómo el software interactúa con los usuarios. Los casos de uso son los más utilizados para modelar el negocio, junto con los diagramas de clases.

La cadena de valor es un modelo que describe las actividades que se llevan a cabo para llevar el producto al cliente final, fue propuesto por el profesor Michael Porter [9]

La cadena de valor es una herramienta de análisis estratégico para la empresa. Su objetivo es maximizar el valor entregado al cliente mientras se minimizan los costos.

Modela las actividades principales de la fabricación del producto o servicio, y las actividades de apoyo. Así como la cadena de suministro.

Con el análisis de la cadena de valor es posible saber en qué punto las tecnologías de información pueden tener un impacto estratégico. Es decir, que hagan más productivo, reduzcan costos o agreguen valor al producto final.

En las actividades primarias se puede modelar logística de entrada, logística de salida, operaciones, marketing y ventas, servicio. En actividades de apoyo: gerencia, administración, recursos humanos, adquisiciones, y tecnología.

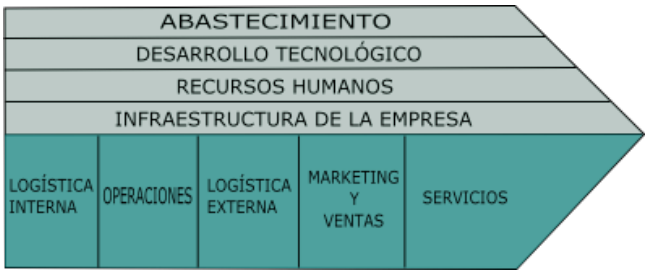

Figura 5. Cadena de valor de Porter
[De Anuskafm, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1641646]

La figura Cadena de valor de Porter, muestra las áreas típicas de una empresa. En las cuales puede interactuar o no el software.

4.1.3 Alinear la cadena de valor con TI

Todo el software que administre el área de TI tiene que ver con algún punto de la cadena de valor. Bien sea en las labores administrativas, de apoyo o el corazón del negocio. El software facilita las operaciones.

Cada software debe modelarse y tenerse en cuenta para las actividades de mantenimiento, cambios, y nuevos aplicativos.

Hay diversas formas de modelar la cadena de valor.

4.1.3.1 Modelado de elaboración propia

Es posible que, entre las estrategias de la organización, esta haya optado por tener modelado propio, que no sigue ningún estándar. Si ya existe, el área de TI puede reutilizar estos modelos para alinear la operación.

La figura Cadena de valor de tienda online, muestra un ejemplo de cadena de valor, de una tienda online.

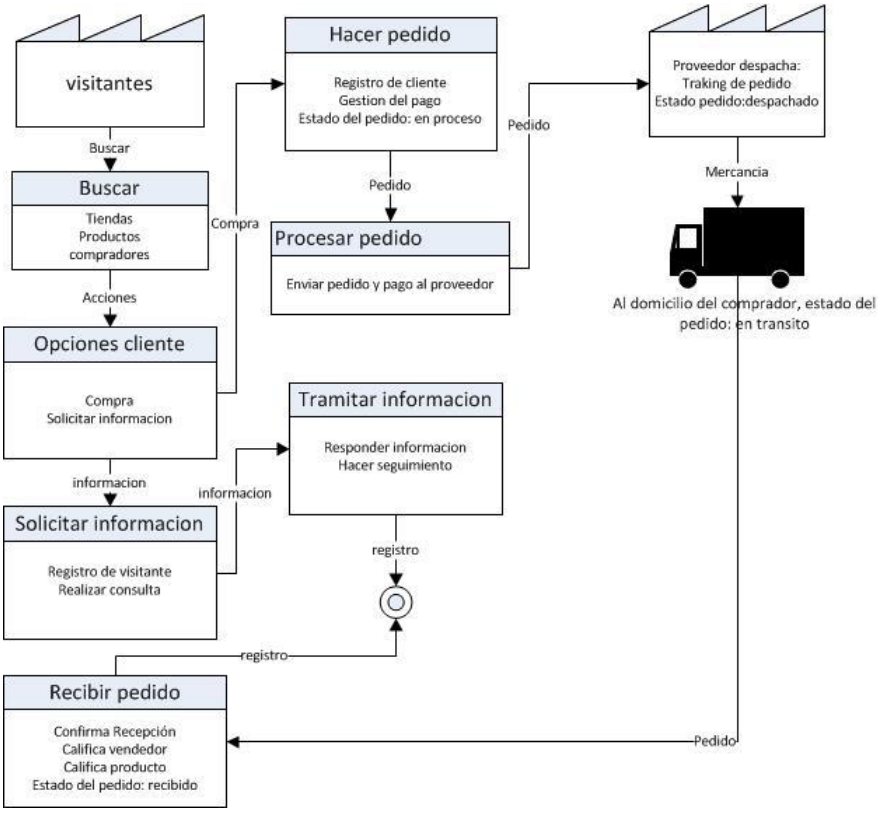

Figura 6. Cadena de valor de tienda online
[10]

4.1.3.2 Modelado UML

El lenguaje unificado de modelado tiene el respaldo del Object Manangement Group, y es el lenguaje de modelado más conocido y usado.

Los diagramas de actividades pueden ser usados para modelar la cadena de valor. También puede complementarse con otros diagramas cómo diagramas de casos de uso.

Este tipo de modelado tiene la ventaja de estar más cerca de la implementación tecnológica. Por lo que su uso es extendido.

La figura Modelado UML de proceso de venta, muestra un diagrama de actividades de un despacho de pedidos

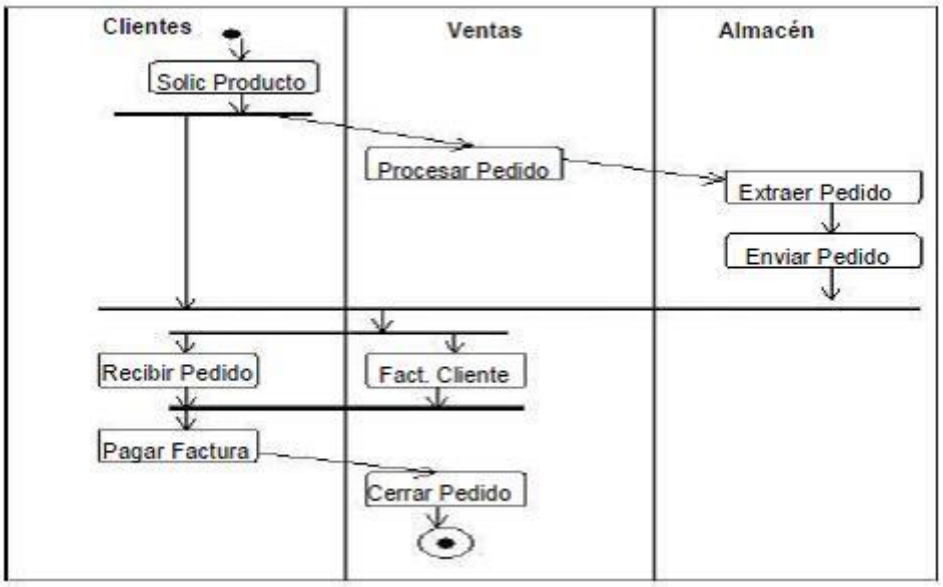

Figura 7. Modelado UML de proceso de venta
[De: http://datateca.unad.edu.co/contenidos/200609/exeuml/leccin_21_diagramas_de_actividades.html]

Es muy importante que el área de TI cuente con un modelo del negocio.

4.1.4 Establecer la estrategia empresarial

La estrategia empresarial es la forma cómo la empresa entrega el producto o servicio al cliente y enfrenta las dificultades del mercado.

Lo que busca es sobresalir en el mercado frente a la competencia, y así ganar nuevos clientes y conservar los actuales. A este sobresalir se le llama ventaja estratégica.

Suele documentarse como la declaración de misión y visión. Y una serie de objetivos para alcanzar la visión cumpliendo la misión.

Principales estrategias

Michael Porter [9] fue el primero en definir las principales estrategias empresariales.

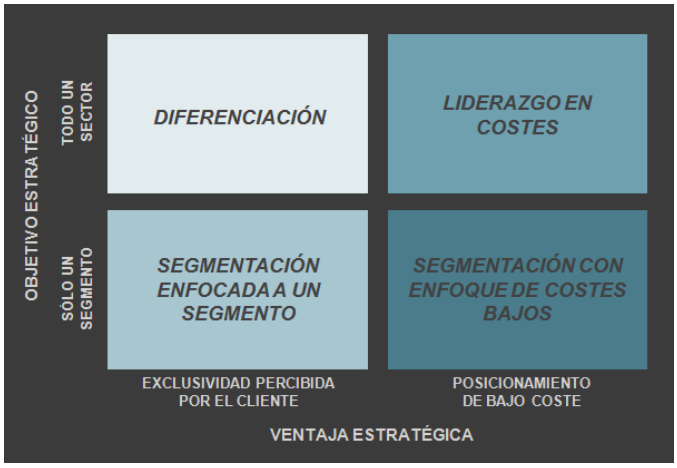

Figura 8. Estrategias genéricas de Porter

[De ManagersMagazine - http://managersmagazine.com/index.php/2010/01/matriz-boston-consulting-group/, GFDL, https://commons.wikimedia.org/w/index.php?curid=9818145]

En la figura Estrategias genéricas de Porter se ven las diferentes estrategias que definió Porter.

La **diferenciación** es la estrategia de las grandes marcas. Muestra que su producto es único, por lo que se justifica pagar un alto precio. Autos de alta gama y bolsos de diseñador tienen esta estrategia.

Liderazgo en costes es la estrategia que busca dar siempre precios más bajos que la competencia. Economías de escala, acceso a mercados complicados, permiso especial de los gobiernos permiten este tipo de estrategia. Es usada por tiendas de descuento y grandes mayoristas.

Segmentación es la estrategia que busca un sector de los clientes. Las editoriales usan esta estrategia para algunos de sus productos literarios: novela negra, novela romántica, libros infantiles. Estos productos son para segmentos específicos de la población.

Segmentación con enfoque en costes bajos. Es la más difícil de las cuatro. Busca ofrecer precios bajos a ciertos segmentos. Es usada por ejemplo por tiendas de ropa especializada: para niños, jóvenes, saldos de temporada.

Las grandes empresas usan combinación de varias estrategias, Apple combina diferenciación con segmentación. No todas las personas con capacidad de compra usan Apple. Por ello se dirigen a cierto segmento amante de la tecnología y con dinero.

Existen otras estrategias y es un tema de investigación constante en ciencias administrativas y empresariales. Estas estrategias se enumeran para mostrar el concepto.

En 4.9 se detalla una estrategia de ejemplo que ha sido constante objeto de estudio en escuelas de negocios. A partir de la cual se genera un cuadro de mando integral.

4.1.5 Socializar la estrategia con TI

El personal de TI debe conocer y apoyar la estrategia organizacional. En muchas empresas, parece que el área de tecnología trabaja en una organización diferente y una de las razones más fuertes es que no conoce la estrategia empresarial.

Es posible que la estrategia no esté documentada, sin embargo, se hace evidente en la forma de hacer negocios o de vender el producto en el mercado.

El modelo de negocio y la estrategia son fundamentales para la implementación del modelo de gobierno de TI propuesto. En 4.9 se detalla cómo usar la estrategia para la construcción del cuadro de mando.

4.1.6 Establecer la estrategia de TI

Como último paso de definir el negocio y la estrategia deben definirse la estrategia de TI.

Esta debe ser planteada mediante objetivos medibles. Y su cumplimiento debe ayudar a alcanzar la estrategia organizacional.

No es una tarea fácil. Para tener éxito en la estrategia hay que tener objetivos que sean realistas, que el personal este consciente de ellos y que haya fuentes de información que permitan medirlo.

El grupo de objetivos estratégicos de TI son el resultado del consenso del equipo y deben quedar almacenado en la base de conocimientos.

4.1.7 Resumen del negocio y la estrategia

Toda organización humana tiene un objeto social que cumplir. Y este es el negocio al que se dedica. Puede ser la elaboración de un producto o la prestación de un servicio. Y puede obtener un beneficio económico o no.

Para llevar el negocio a los consumidores hay un proceso, ese proceso es el modelo de negocio o cadena de valor. Y en varios puntos del modelo de negocio juega un papel el software. El fin último del software es hacer más productivo el negocio. Automatiza tareas repetitivas, lleva registros o impulsa las tareas.

La estrategia es la forma cómo la organización compite en el mercado y busca maximizar los beneficios.

Es deber del área de tecnologías de información, conocer el negocio, el modelo de negocio y la estrategia de la empresa. Estos son la base sobre la cual trabajar la estrategia del área de TI que debe ayudar a alcanzar la estrategia organizacional. Los cual se gestiona con un cuadro de mando.

4.2 PROCESO GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE

Figura 9. Pasos para establecer administración de configuración del software [Elaboración propia]

El principal objetivo de la gestión de configuración del software es manejar con eficiencia los cambios que realiza un equipo de desarrollo al conjunto de programas. Cualquiera que sea la tecnología o metodología de desarrollo empleada, debe gestionarse la

configuración. En pocas palabras la gestión de la configuración son las herramientas: como directorios, repositorios, compiladores, servidores de pruebas y producción y las reglas para su uso.

Con reglas claras el equipo no perderá el tiempo en buscar el código, los ejecutables, la última versión… Sino que irá directamente a donde debe ir.

Los marcos de procesos sugieren usarla como una actividad de apoyo. CMMI [1], ISO, ITIL [11], IEEE [12]… Tienen entre sus prácticas administración de la configuración. Sin embargo, aplicar estos conceptos al trabajo diario no es fácil. Y más que apoyo pueden llegar a ser un obstáculo para el equipo de desarrollo.

En la presente memoria se propone el uso de patrones de diseño, concepto ampliamente usado arquitectura y desarrollo de software.

Son los ingenieros Stephen Bercsuk y Brad Appleton, quienes proponen en su libro Software Configuration Management Patterns: Effective Teamwork, Practical Integration [13] el uso de patrones para solucionar problemas de administración de la configuración.

Los patrones de diseño, con soluciones eficaces a problemas conocidos. En esta memoria se presentan patrones de diseño, su propósito y una breve explicación para su implementación. Con estos patrones se cubren la practicas de la mayoría de marcos de proceso y planes de calidad.

Que hace la gestión de la configuración del software.

Se debe diferenciar lo que es parte del producto software, de lo que es parte administrativa. Como cronogramas, presupuestos o listas de personal.

La gestión de la configuración propuesta en el presente capitulo, se refiere a la administración del producto: código fuente, esquemas de bases de datos, software para construir, compilar y probar el producto y documentación del mismo.

En la documentación se debe incluir la configuración básica. El tipo de usuarios y perfiles necesarios, y el software base sobre el que funciona la aplicación.

No se debe confundir la gestión de la configuración con la gestión de activos. La primera es una actividad de apoyo al proceso de desarrollo, y la segunda es una actividad de gobierno de los recursos.

Teniendo esto es cuenta, la gestión de la configuración, en adelante CM.[1], debe permitir:

- El equipo puede trabajar junto y compartir modificaciones a los mismos elementos.
- Se puede compartir el esfuerzo de modificar un mismo modulo o incluso de los mismos archivos por varias personas.
- Cualquier miembro del equipo debe poder acceder a una copia estable de los elementos almacenados. Por ejemplo, para corregir un error propio o de otro miembro del equipo que no esté disponible.
- Se pueden “devolver” a la versión del día anterior o de cualquier otro día de los elementos almacenados.

Conseguir esto permite actividades como:

[1] Por Configuration Management

- Desarrollar la próxima versión del software y solucionar los problemas del actual.
- Compartir código entre varios miembros del equipo de una forma controlada.
- Identificar qué se modificó en cada versión: qué se añadió, qué se eliminó.
- Analizar en qué momento se modificó un elemento y qué se hizo.

Un buen sistema de CM debe permitir estas y otras actividades de una forma natural al trabajo diario. Si en el proceso hay mucho que recordar también habrá mucho que se olvidará.

Que no hace la gestión de la configuración del software.
CM trae muchas ventajas en su uso, pero no es algo que deba:

- Realizar la verificación/validación del software construido: si lo que se desarrolló cumple o no los requisitos del cliente no es responsabilidad de CM verificarlo.
- Ser un reemplazo de la administración del área, CM no debe decir a los desarrolladores o al equipo en general que hacer o cómo hacerlo.
- Una metodología de desarrollo: CM apoya el modelo de desarrollo, pero no interviene en las etapas o la forma de organizar el desarrollo. CM se adapta a la metodología empleada y no al revés.

Confundir lo que debe hacer CM puede dar como resultado un proceso burocrático e inútil.

4.2.1 El entorno de desarrollo

Para usar CM correctamente, se debe saber cómo sus técnicas encajan en el proceso de desarrollo. Para ello hay que tener en cuenta ciertos aspectos.

4.2.1.1 Principios

CM es el conjunto de actividades que se usan para crear y mantener un conjunto de componentes de software que se ejecutan en un entorno productivo.

El control de versiones es el componente CM que los desarrolladores verán con frecuencia. Para que sea útil, se deben cumplir ciertas reglas.

En general las políticas que deben seguirse con independencia de las herramientas son:

- Usar el control de versiones: este es el sistema por el cual las organizaciones comunican su trabajo. Esto parece obvio pero muchas organizaciones no lo hacen, incluso algunas que tienen herramientas implementadas no las usan correctamente, simplemente intercambian archivos sin tener control de los cambios que se realizan.
- Integrar con frecuencia. Tener algún tipo de proceso de compilación periódica. Cuanto más tiempo se deje de integrar, mayor probabilidad de problemas habrá.
- Permitir el trabajo autónomo. En cualquier momento un miembro del equipo debe poder tener su propia línea de desarrollo, compartir archivos es necesario, pero CM debe permitir la independencia.
- Usar herramientas. Si hay demasiados procesos manuales, las personas cometen errores. Todo lo que se pueda hacer con una herramienta debe ser realizado por esta.

Una reflexión acerca de qué es software desde el punto de vista CM

En resumen, software es el conjunto del código fuente. El equipo escribe código, y este genera un programa que hace algo útil. Otros artefactos como librerías, bases de datos, documentación y el entorno de implementación hacen que el software funcione. Pero es el código el que define como trabajan estos otros artefactos, sin código estos artefactos son inútiles.

Sin embargo; el proceso de desarrollo es más que escribir código, las políticas corporativas, la forma de contratar el personal, la seguridad que se implemente influirá en mayor o menor medida en la forma en que se implemente el sistema de CM. E influirá en el programa construido.

Un proceso CM que no tome en cuenta el entorno en el que debe trabajar está condenado al fracaso. El código fuente es el corazón del sistema de control de versiones, pero no es lo único que debe tomarse en cuenta.

4.2.1.2 El espacio de trabajo privado

El desarrollo sucede en el ordenador del desarrollador. El espacio de trabajo privado es el conjunto de componentes que el desarrollador usa para modificar y ejecutar el programa que está construyendo.

Un desarrollador puede tener uno o más espacios de trabajo, dependiendo de los proyectos que esté trabajando.

En el patrón Espacio de trabajo privado se detalla más de este tema.

4.2.1.3 Arquitectura

Hay discusiones acerca de lo que es la arquitectura de software. No está en el alcance de este documento discutir lo que es. Para CM la arquitectura es la que define como las piezas del software encajan entre sí y cómo se pueden sustituir. Y esa es una pieza clave en la organización del repositorio.

El código es el que impone la forma de organizar estos componentes ya que es la forma como trabajan correctamente.

Se debe tener en cuenta esta arquitectura a la hora de organizar la estructura de carpetas del sistema de control de versiones.

4.2.1.4 La organización

Aspectos corporativos deben ser tenidos en cuenta a la hora de organizar el proceso CM.

La distancia de los desarrolladores: pueden estar todos conectados a la red LAN de la empresa, o pueden trabajar desde sus casas a través de Internet. Esto influye en las políticas de seguridad que debe tener el control de versiones: solo LAN. VPN, Internet por ip pública…

Cultura organizacional, no es igual un equipo que se comunica por chat que uno que solo lo hace a través de comunicaciones formales por correo electrónico. Si los cambios

se hacen con historias de usuario o con complejos modelos UML. CM debe formar parte de ese sistema de comunicación.

Jerarquía. Se debe tener claro quien tiene el poder de tomar decisiones, hay que recordar que CM es un proceso de apoyo, no es decisorio. Tener claro quien toma decisiones es de vital importancia para el proceso.

Estas y otras características propias de la organización se deben tomar en cuenta para que el proceso CM encaje dentro de la organización.

4.2.1.5 Juntándolo todo

Es fácil perderse entre políticas, tecnologías, arquitectura, herramientas y terminar con un proceso CM inaplicable que no cumpla la función última que debe: ayudar en la construcción de software.

El objetivo no es, por ejemplo, crear ramas. Las ramas sirven para aislar un proceso de desarrollo, o de lograr el desarrollo concurrente pero no es el objetivo final, la organización se puede ver abrumada con un proceso CM que no apoye, sino que se convierta en un verdadero cuello de botella.

Para juntar todo esto se propone el uso de los patrones que se describen a continuación, soluciones eficientes a problemas que se pueden reproducir con facilidad.

4.2.2 El control de versiones

Después de tener claro cómo funciona el software y la organización se deben implementar un control de versiones. Este, es un software que controla los cambios que se realizan al código fuente y a otros documentos. Existen opciones de pago y de código abierto, pero todas manejan más o menos las mismas funcionalidades: almacenamiento de código fuente, ejecutables y otros elementos de trabajo. Control de los cambios hechos a estos con fecha usuario y tipo de cambio realizado. Y se ofrece la posibilidad de ver una versión anterior del componente.

El directorio debe organizarse de acuerdo con cómo lo use el sistema que construye el software. De modo que un nuevo desarrollador simplemente obtenga la última versión estable del código fuente y construya el software.

4.2.3 Patrones

Para que CM ayude a trabajar con eficacia se debe pensar en cómo todas las etapas de desarrollo interactúan entre sí. Una forma de modelarlo es pensar en el proceso de desarrollo en términos de las relaciones entre los patrones del entorno de desarrollo.

4.2.3.1 ¿Qué son los patrones?

Cristopher Alexander es el pionero de los patrones, o lenguaje de patrones. Arquitecto de profesión. Se dio cuenta que existían problemas que se repetían una y otra vez en su medio; y que se podía emplear la misma solución una y otra vez.

A estas soluciones las llamó patrones. Pero el patrón no es solo una solución, es una solución eficiente al problema. Así se dio la base de soluciones a problemas de arquitectura y urbanismo que aún se usan.

Aunque el trabajo de Alexander fue sobre arquitectura pronto se amplió a otras disciplinas. En ingeniería de software se usan, ya que el software trata de construir cosas similares a otras cosas que se hicieron en el pasado. El libro Patrones de Diseño [14], catalogó una serie de patrones aplicados a la programación orientada a objetos. Aporta soluciones eficaces a una serie de problemas de programación.

Se han escrito otros libros sobre patrones de software, sobre arquitectura, incluso sobre temas de organización del personal.

Los patrones en CM son el intermediario entre el equipo de trabajo y la arquitectura de software. De cómo el espacio de trabajo privado de los desarrolladores, analistas, diseñadores, analistas de pruebas. Interactúa con el sistema de control de versiones.

4.2.3.2 Patrones en CM

Los patrones son una forma muy efectiva de pensar en CM por las siguientes razones:

- CM implica la forma de trabajo: la manera como se construye el software.
- CM absorbe los procesos de ingeniería: Análisis, diseño, desarrollo y pruebas o los que apliquen. Y los elementos resultantes: Documentación, código fuente, ejecutables, etc.
- Los estándares de CM (CMMI, ITIL, ISO. IEEE) están llenos de “buenas prácticas”. Pero para aplicarlos eficazmente se debe comprender como las prácticas implementadas se relacionan con las de los otros procesos.
- Pequeños cambios en las prácticas de CM redundan en enormes beneficios en el proceso.
- Los procesos y los métodos son dinámicos, y es difícil modelarlos, es más fácil modelar estructuras estáticas y después definir su comportamiento.

Los procesos pueden ser ejecutados por las personas, pero es mejor si se hace de una forma intuitiva, si son impuestos por las herramientas. De lo contrario las personas los verán de una forma arbitraria e inútil.

Los patrones dan una forma de ver los procesos y los problemas de otra manera. La forma como los patrones se relacionan entre si es tan importante como el problema que resuelven.

También existe la posibilidad que no se discuta una solución porque parece obvia, pero la forma de aplicar la solución no es tan evidente, los patrones dan esa oportunidad de hablar de ese problema que parece tan infantil, pero que si se queda sin solucionar ocasionará grandes problemas.

4.2.3.3 Estructura de los patrones que se muestran en el presente estudio

Los patrones usados en este trabajo se basan en los que se exponen en el libro: Software Configuration Management Patterns [13], y se sigue la estructura que se propone en el libro:

- Título del patrón: un nombre y una descripción general del patrón.
- Una imagen: que puede servir de metáfora del mundo real, son imágenes libres de Internet que buscan dar una idea del patrón.

- Contexto: cuando se debe tener en cuenta el uso del patrón.
- Problemas que resuelve. Exposición de los problemas que resuelve el patrón.
- Detalle: solución del patrón.

Problemas no resueltos: problemas y posibles soluciones a través de otro patrón. Los que se requieren para CM se detallan a continuación.

4.2.4 La línea principal

La línea principal de desarrollo es la que contiene el software estable, el que ha superado las revisiones técnicas y es la base para los próximos desarrollos, es: "la última liberación"

Patrón

Título: Línea principal de desarrollo (mainline)

Figura 10. Línea principal de Shinkansen Japón
[de: www.pixabay.com]

Contexto:
¿Cómo mantener un conjunto manejable de líneas activas de código evitando el crecimiento del árbol de desarrollo?
¿Cómo minimizar la fusión?

Problemas que resuelve:
Contar con un sitio común, donde ir a buscar la última versión del código.

Poder realizar ramificaciones de esta línea principal para aislar los desarrollos. Es decir; que cada desarrollador pueda modificar código sin modificar el trabajo de otros.

Continuar el trabajo inmediatamente se termina o se realiza la liberación.

Detalle:

Cuando se desarrolla, desarrollar fuera de la línea principal. Estas líneas "aparte" son copias de la línea principal[2]. La línea principal mantiene a salvo el software mientras se desarrollan los cambios.

Una vez el cambio se termina se fusiona de nuevo con la línea principal.

Considerar la estrategia global antes de ramificar. En caso de duda ir por un modelo más simple.

La razón para una línea principal es tener una fuente de código central para las ramas y las fusiones resultantes.

La línea principal se inicia con la última versión o con una versión del software. Si el desarrollo es nuevo, hay una sola línea de desarrollo y esta es la línea principal por definición.

No se debe crear una rama si no se tiene una razón para ello. Se debe evaluar el esfuerzo de fusionar con la línea principal. Se deben usar buenas prácticas de desarrollo que garanticen que la modificación de la rama cumple los requisitos de calidad y que no arruinará la línea principal después de fusionar.

La línea principal no tiene que ser la raíz del sistema de versiones, puede ser una rama. Lo que se debe tener claro es que en algún momento se integrará en una sola línea de código.

Cuando se termina un desarrollo, para iniciar el siguiente, primero se debe fusionar el desarrollo anterior con la línea principal, así, esta línea principal será la base para el nuevo desarrollo en una rama a partir de ella.

Se puede tener un modelo de línea principal como el de la figura:

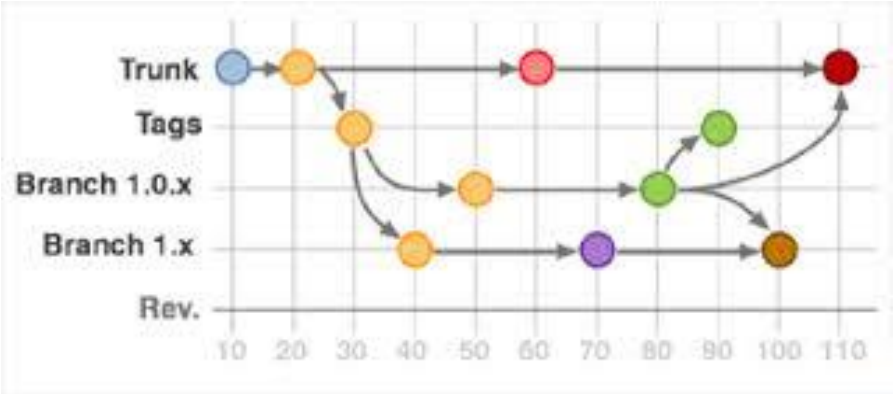

Figura 11. Línea principal en el trunk
[de https://es.wikipedia.org/wiki/Archivo:SVN_Branches_and_Trunk.gif#filelinks]

Este modelo propone que la línea principal es el directorio Trunk, de allí se deriva un Tag, y de este se derivan las ramas (branch) para cada una de las versiones. Finalmente, el primer branch se une a la línea principal, y el segundo branch aún continúa su evolución y en él se han unido los cambios del primer branch. El objetivo de este modelo es siempre fusionar en el trunk.

[2] Este proceso se le llama ramificación, en inglés: branching. Son copias de la línea principal donde se realizan cambios sin afectar la línea principal.

Para implementar este patrón se pueden seguir los siguientes pasos:

1. Crear un directorio con el código actual como punto de partida
2. Comprobar que tiene todos los cambios en este directorio
3. Seguir siempre los procesos de verificación antes de integrar un nuevo cambio, así se mantiene esta línea de código útil.

Problemas no resueltos:

Una vez se tiene la línea principal activa, es necesario resolver como mantener la línea principal activa y útil cuando muchas personas están trabajando en el desarrollo. El patrón: línea activa de desarrollo, describe cómo administrar eso.

4.2.5 línea activa de desarrollo

La línea activa de desarrollo es la parte del repositorio donde se conserva el código que se está trabajando y que es útil. Es allí donde los desarrolladores conectan su espacio de trabajo y almacenan el código que no tiene errores de compilación ni de integración.

Es un código en constante evolución. Con él se espera no perder la estabilidad del producto y la codificación concurrente.

Patrón.

Título: línea activa de desarrollo (line active development)

Figura 12. Edificios residenciales en construcción
[Diario El País, Cali Colombia]

Contexto. ¿Cómo mantener una línea de código en constante evolución lo suficientemente estable para ser útil?

Problemas que resuelve.

Se tiene un grupo de personas que trabajan de forma concurrente. Cuanta más gente trabaja en el código más comunicación se hace necesaria y también es mayor la probabilidad de tener conflictos.

Si se piensa en el desarrollo de software como un trabajo concurrente, hay que verificar el código cada vez que se registre un cambio en el repositorio. Si un cambio arruina el trabajo de otro puede causar retrasos a todo el equipo. Aun si se ha probado, ese código que se sube podría ser no compatible con lo que se registre momentos después.

Si se pone demasiado esfuerzo en comprobar cada cambio con un conjunto exhaustivo de pruebas, puede causar retrasos.

Pruebas de larga ejecución pueden tener resultados que a la larga resulten contraproducentes, como se ve en la figura.

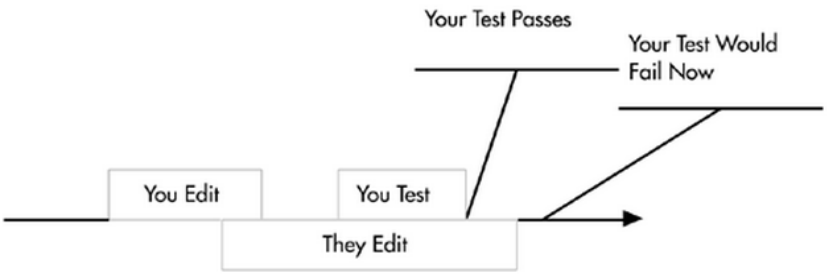

Figura 13. Pruebas de larga ejecución [13]

Cuando terminan las pruebas ya se ha registrado un nuevo cambio al que no se le han hecho pruebas. Hacerlas de nuevo producirá nuevos retrasos.

Si se define que solo una persona haga cambios a la vez, para ejecutar las pruebas puede generar un lento progreso, como se ve en la figura.

Figura 14. Código estable, pero de lento progreso [13]

Si se toma el extremo que, aunque las pruebas se ejecuten mal, continúe el desarrollo se tendrá una línea de código inútil.

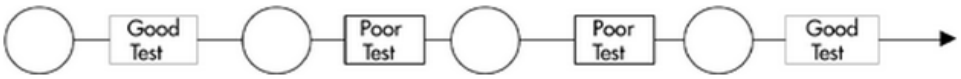

Figura 15. Línea de código activa, pero inútil [13]

También es posible que la arquitectura sea modular de modo que difícilmente un cambio afecte a otro modulo, pero aun así es posible que dos personas deban cambiar el código de un mismo archivo.

Lo que se requiere es equilibrio, código aceptablemente estable, y desarrollo que evolucione constantemente.

Detalle:
Instituir políticas eficaces para hacer que el código de la línea activa sea lo suficientemente estable para ser útil. No se trata de una línea de código perfecta, sino lo suficientemente estable y activa.

Una línea de desarrollo activa tiene cambios frecuentes y puntos de control para probar.

La parte más difícil es encontrar la "mejor manera" para la línea de código. Se debe proceder de una manera similar al análisis de requisitos. Los clientes quieren perfección rápida y barata, esto no es posible, hay que ir en busca de las respuestas de:

- ¿Quién usa la línea de código?
- ¿Cada cuánto y cómo se libera el producto?
- ¿Qué tipos de pruebas se realizan?
- ¿Qué cantidad de desarrollos son concurrentes?
- ¿Cuáles son los costos de los errores?

Por ejemplo, para un equipo grande en un desarrollo nuevo, cierta inestabilidad es soportable, ya que la prioridad es la velocidad. Si la liberación está cerca en el tiempo esta inestabilidad no es aceptable. Es necesaria más verificación.

Si hay un buen sistema de pruebas automatizadas, una copia del mismo en los equipos de los desarrolladores puede ser una solución para que se aplique el set de pruebas cada vez que se considere que hay un gran cambio.

Si el sistema de pruebas tiene muchos procesos manuales. El proceso de pruebas debe aplicarse con versiones más grandes.

En cualquier caso, la mayoría de herramientas de control de versiones soporta la automatización de ejecución de pruebas. La frecuencia con que se ejecuten es la decisión que se debe tomar teniendo en cuenta el justo equilibrio entre calidad del producto y la velocidad de desarrollo del mismo.

Problemas no resueltos.
Los desarrolladores deben ser capaces de identificar cual es la línea de desarrollo activo. La política de líneas de código resuelve este problema.

También se requiere aislamiento del trabajo de los desarrolladores, por lo que el espacio de trabajo privado ayudara a solucionar este problema.

Posible Implementación
La forma de implementar este patrón es a través de la política de ramificación. Las ramas son derivaciones de la línea principal.

Por ejemplo, al tratarse de un producto en continuo mantenimiento al que se le van a hacer cambios frecuentes y cortos, se puede tener una sola rama a la que conectan todos los desarrolladores su espacio de trabajo.

Si hay un cambio que va a llevar mucho tiempo, es buena idea tener una rama separada para ese cambio largo.

Una política de ramas puede ser así:

1. Para los cambios cortos todos los desarrolladores se conectan a la rama “desarrollo continuo”
2. Para los cambios largos (más de dos semanas) se creará una rama separada que debe actualizarse con los cambios de la rama desarrollo continuo
3. A la rama solo puede subirse código útil, es decir; que compile y pueda implementarse
4. Se hará una revisión automatizada de la rama desarrollo continuo cada noche: compilación y despliegue automatizado, se reportará al equipo los defectos encontrados los cuales deben ser corregidos antes de subir el siguiente cambio.

4.2.6 El espacio de trabajo privado

En la línea activa de desarrollo uno o más desarrolladores hacen cambios al código. Hay que estar seguro de que se trabaja con la última versión.

El espacio de trabajo privado es el refugio de un producto de trabajo que está en constante cambio. Es el ordenador del desarrollador.

Patrón.

Título: Espacio de trabajo privado (Private Workspace)

Figura 16. Espacio de trabajo [www.pixabay.com]

Contexto.
¿Cómo mantenerse al día con una línea de código que cambia constantemente? ¿Cómo avanzar sin ser distraído por los cambios que ocurren alrededor?

Problemas que resuelve.
Los desarrolladores necesitan un lugar donde puedan guardar su código aislado de los cambios mientras terminan una tarea.

En un equipo de desarrollo de software la gente trabaja en paralelo, así se espera que el desarrollo sea más rápido que si se trabajara individualmente. Pero se tiene el problema de la gestión de estos cambios paralelos.

Existe un conflicto entre mantenerse al día con los cambios y trabajar en un ambiente con un cambio mínimo.

Existen dos tareas en el desarrollo de software que se realizan en equipo:

1. Construir y probar los cambios propios.
2. Integrar el código propio con los cambios de los demás.

Hay dos alternativas extremas: integración continua, y retrasar la integración.

Con la integración continua se integran los cambios de forma inmediata que se suben al repositorio, pero puede tomar mucho tiempo esta integración, lo que hace difícil detectar una falla. Una falla puede producirse por uno o más cambios o por la combinación de varios cambios.

Con la integración al final se realiza la integración en el último momento posible. Esto hace más sencillo el trabajo, pero se pueden tener muchos problemas al final.

También puede suceder que se trabaje, no en la última versión del código, sino en "una versión" debido a un desarrollo de muy larga data. Lo que significa que la integración con la última versión puede dar muchos problemas.

Se necesita una manera de controlar los cambios sin estar demasiado lejos de la línea principal de código.

Detalle

Hacer el trabajo en un espacio de trabajo privado, donde se tenga el control de las versiones del código y componentes de lo que se está trabajando.

Cada miembro del equipo debe poder crear un espacio de trabajo con una versión estable del software. Un espacio de trabajo es: una copia de las versiones correctas, de los archivos correctos en los directorios correctos. También es el lugar donde se realizan cambios hasta que se comprueba que son correctos.

Un espacio de trabajo privado puede contener:

- El código fuente.
- Cualquier componente de construcción local
- Objetos de terceros que no se quiera o no se puedan construir
- Los ejecutables del software. Bien sea construidos por el mismo desarrollador o descargados del repositorio con la versión correcta.
- Los datos, y acceso a los mismos para ejecutar el sistema
- La información que identifica las versiones de todos los componentes del sistema
- Secuencias de comandos para construir el sistema.

No debe contener:

- Versiones privadas de los comandos que hacen cumplir las políticas
- Componentes que se encuentran en el control de versiones, pero copiados de cualquier otro lugar

- Cualquier herramienta diferente a los demás miembros del equipo. Los compiladores, editores intérpretes deben ser los mismos para todos.

Para el trabajo diario, se puede seguir pasos similares a este:

1. Ponerse al día. Si se trabaja en una rama privada se debe estar seguro de que se tienen los últimos cambios de la línea activa de desarrollo.
2. Hacer los cambios. Realizar los cambios que se requieran.
3. Hacer una construcción privada. Compilar los componentes, web service… Que se hayan modificado, ponerlos en el lugar correcto en el espacio de trabajo privado.
4. Hacer una prueba unitaria. Este tipo de prueba garantiza que las cosas funcionen como deberían funcionar.
5. Actualizar el espacio privado con los cambios de todos los demás.
6. Reconstruir. Construir de nuevo los componentes y hacer una prueba de humo para verificar que nada se dañó.

Uno de los riesgos que se corre es que los desarrolladores trabajen con código conocido demasiado tiempo. Y que en un momento dado trabajen con código obsoleto. Por ello es mejor trabajar pequeños cambios que se integren en cortos periodos de tiempo y que se actualicen los espacios privados periódicamente.

También se puede promover la reconstrucción periódica del espacio de trabajo privado para evitar los posibles errores de la herramienta de gestión de versiones.

Problemas no resueltos
Una vez se tenga el espacio de trabajo privado aún se pueden introducir errores, por lo que un sistema privado de construcción ayuda con este problema.

Es necesario tener acceso a un repositorio que contenga el código fuente y los componentes relacionados.

Una vez terminado el desarrollo se debe enviar a la integración centralizada.
Implementación
Una implementación posible es la conexión del espacio de trabajo con la línea activa de desarrollo. A partir de esta se construye por primera vez en el equipo local los componentes y se inicia el desarrollo de las modificaciones. Luego seguir los pasos del patrón de diseño especificados en detalle.

4.2.7 El repositorio de recursos

Para construir un espacio de trabajo privado, o para el sistema de integración, se requieren las versiones correctas de los componentes. Se debe contar con un sitio desde donde tomar estos componentes con facilidad.

Patrón
Título: Repositorio (Repository)

Figura 17. Almacén de amazon.com
[De: http://www.buzzfeed.com/mjs538/what-it-looks-like-inside-amazoncom]

Contexto

¿Cómo obtener las copias correctas de los componentes para un espacio de trabajo privado?

Problemas que resuelve.

Como se analizó todo desarrollo se realiza en un espacio de trabajo privado; en el ordenador de cada desarrollador.

La creación de este espacio de trabajo debe ser fácil y repetible. Y para construirlo se necesitan una serie de componentes: código fuente, servidores web, librerías, compiladores, editores de código, programas cliente del sistema de versiones etc.

Cada persona debe saber dónde están estos componentes, se pueden tener centralizados en una sola ruta o en un conjunto de directorios al alcance de todos.

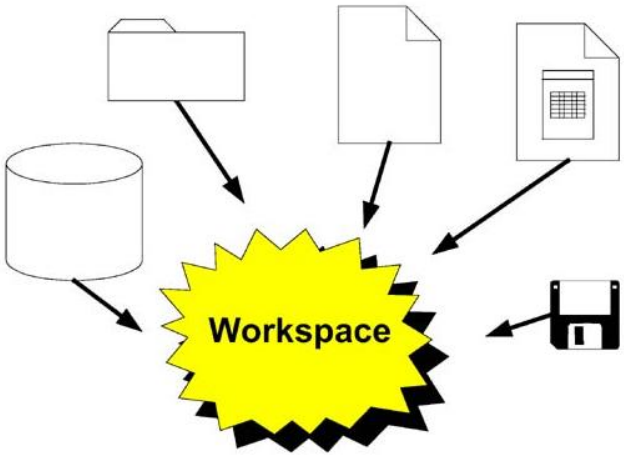

Figura 18.Componentes de un workspace
[13]

Algunos componentes tienen orígenes obvios, el código fuente viene del sistema de control de versiones. Scripts de instalación de bases de datos también pueden estar allí.

Se agregan otros componentes en un servidor: Compiladores, IDE, pero se debe tener en cuenta que copiar archivos de múltiples orígenes deja espacio para el error.

Es posible crear una herramienta para la administración de los espacios privados, pero aun así hay que mantenerla al día con las nuevas ubicaciones y políticas. Esa carga de mantenimiento de la herramienta puede distraer al equipo de su trabajo.

Se puede llevar una lista de los componentes y mantenerla actualizada, pero un desarrollador puede estar en medio de una tarea cuando recibe la notificación de actualización y la deja para más tarde, y se olvida de realizar la actividad.

Otra cuestión es regresar a una configuración en particular en el tiempo, volver a actualizar el espacio de trabajo según como estaba en la versión xx del software.

Detalle

Tener un único punto de acceso, un repositorio, para el código y artefactos relacionados. Hacer la creación de un espacio de trabajo del desarrollador tan simple y transparente como sea posible.

Hay que hacer que el trabajo de crear un espacio privado sea sencillo y repetible. Hay que tener la capacidad de crear un espacio de trabajo privado que contenga los elementos de cualquier revisión del producto, incluyendo los componentes de terceros y los artefactos construidos. El mecanismo también debe ser fácil de identificar si hay una nueva versión de un elemento existente o un nuevo componente que se necesita cuando se está trabajando en un desarrollo. La figura muestra esto.

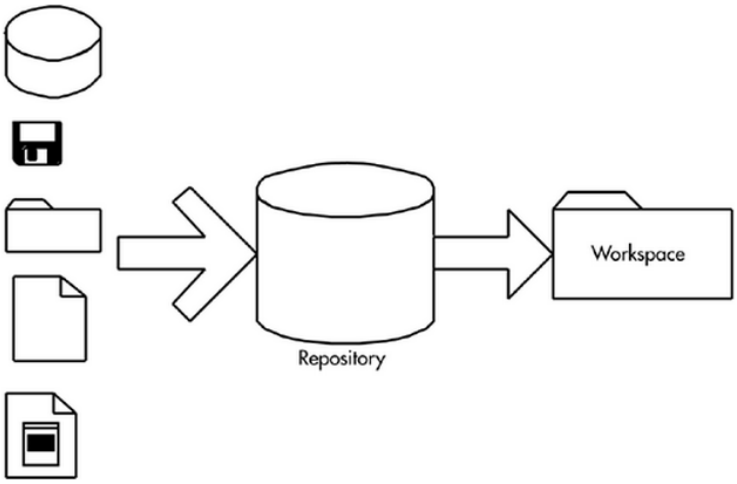

Figura 19. Poner los componentes en un repositorio [13]

Hay muchas maneras de implementar este patrón. La más simple es copiar todos los elementos necesarios en el sistema de control de versiones. Pero es un desperdicio enviar al sistema de copias de seguridad, por ejemplo, versiones de compiladores que no van a cambiar en mucho tiempo. Todo esto depende del sistema de control de versiones usado, el entorno de compilación y en cierta medida la cultura del equipo.

Se deben cumplir tres requisitos para su aplicación:

1. Debe ser fácil de usar y repetible.
2. Debe contener todo lo necesario para crear un espacio de trabajo privado, para un estado particular del proyecto incluyendo scripts y objetos construidos.

3. Se debe trabajar en todas las versiones del proyecto.

Al usar el sistema de control de versiones, este debe reflejar la estructura de construcción del software, el etiquetado (tags) marcará el estado de los componentes en cada versión. Y la línea principal tendrá “lo último” así, si se tiene conectado el entorno local con el sistema de control de versiones, el comando update traerá la última actualización, el comando switch cambiará a la versión que se requiera.

Apuntar cada directorio requerido del espacio de trabajo a un directorio del repositorio facilita el trabajo.

La figura muestra esta situación:

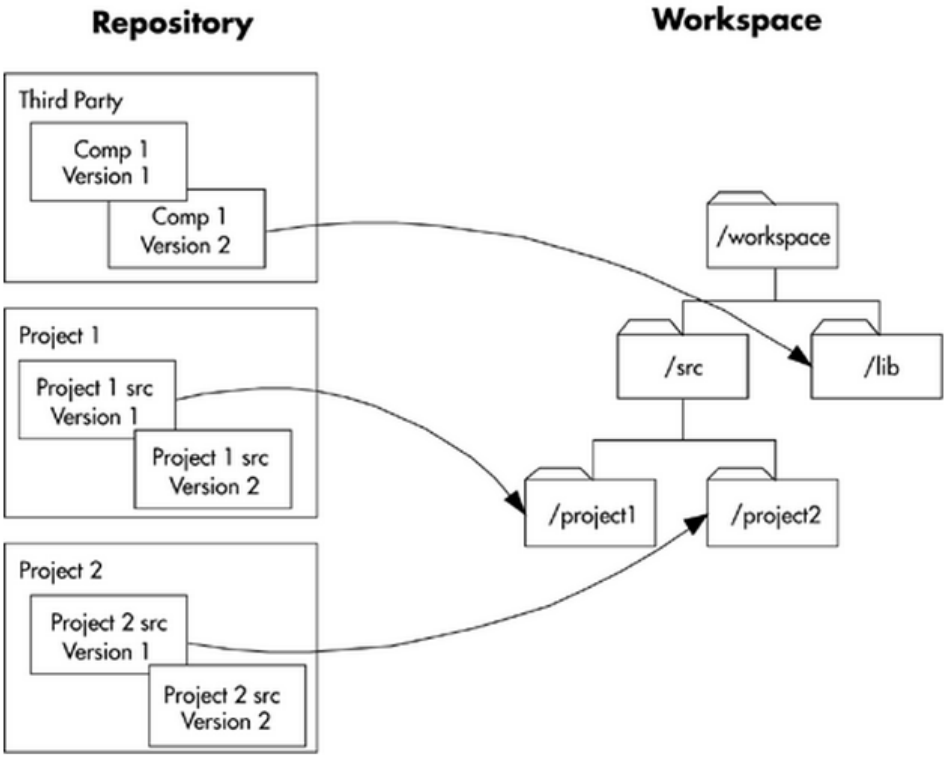

Figura 20. Directorios del repositorio y su relación con el espacio de trabajo privado [13]

De esta manera el sistema de control de versiones refleja el entorno de compilación y la historia de los cambios puede ser rastreado.

Problemas no resueltos

Organizar el código de terceros

Implementación.

Una de las mejores formas de implementar este patrón es por medio de un tutorial para construir el espacio de trabajo local, bien sea en video o escrito, debe tener las instrucciones de como instalar un espacio, que permita construir y modificar el software en el computador personal del desarrollador.

Todo lo necesario está en un único repositorio del que es muy fácil extraer las versiones correctas de los componentes.

Un ejemplo de contenido:

Usuarios y permisos requeridos
De dónde y cómo descargar los IDE de desarrollo, compiladores y librerías necesarias
De donde descargar el código fuente (línea activa de desarrollo)
Como configurar todo para que funcione.

4.2.8 Establecer Integración (manual o continua)

Todos los desarrolladores trabajan en sus espacios de trabajo privado y deciden cuando tomar los cambios de otros en sus desarrollos. Pero varias personas pueden estar haciendo cambios que deben entregarse juntos en el producto terminado. Este patrón se asegura que el código siempre se acumule.

Se busca que se tenga un lugar centralizado para acumular el código útil de todos los desarrolladores y que debe entregarse junto. Esto se suele llamar una versión[3].

Patrón

Título: Integración Centralizada (build integration)

Figura 21. Ensamblaje de un Lamborgini
[De: www.diariomotor.es]

Contexto:
¿Cómo asegurarse que el código siempre se acumula de forma fiable?

Problemas que resuelve
Debido a que varias personas hacen cambios al software. Ningún desarrollador puede estar cien por ciento seguro que un cambio se hará siempre después de que el cambio anterior se haya unido a la línea principal. Alguien puede hacer un cambio paralelo que

[3] En inglés release

es incompatible con otro cambio. La comunicación puede ayudar, pero los problemas siguen sucediendo.

Cuando se suben cambios al control de versiones a pesar de las mejores intenciones se pueden subir errores. Puede ser que la versión del compilador o de una dependencia sea diferente.

Una solución es que cada desarrollador construya todo el sistema. Pero construir todo puede ser algo que tome mucho tiempo, y tiempo es algo que el desarrollador no puede desperdiciar. Pero la integración puede ser algo muy complicado. Ya que las piezas puede que no encajen como se ve en la figura.

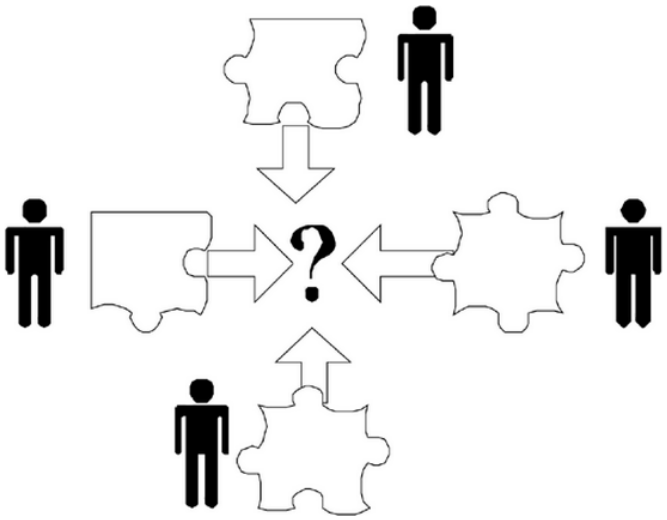

Figura 22. Integración difícil
[13]

Localizar el error es un trabajo frustrante, se debe tener un sistema que garantice que las inconsistencias son encontradas lo más rápido posible y de forma centralizada y automatizada.

Detalle.
Asegurarse que todos los cambios y sus dependencias se construyen utilizando un proceso de generación de integración centralizada

El proceso de construcción debe cumplir las siguientes condiciones:

- Reproducible
- Construir lo más cerca posible al producto final. Pequeños componentes como las etiquetas de versión pueden variar. Pero es mejor si la integración es la misma que un producto entregable. Al final de la integración se debe tener listo un producto para pruebas.
- Automatizado. Entre más larga sea una compilación, hay mayor probabilidad que los equipos cometan errores. Si el sistema de control de versiones es compatible con desencadenantes programar la construcción después de cada cambio.
- Notificar. Debe tener un mecanismo para registrar las inconsistencias. Entre más rápido se identifican errores de generación, más rápido se pueden corregir.

Además, la notificación debe permitir identificar el cambio que arruinó la construcción.

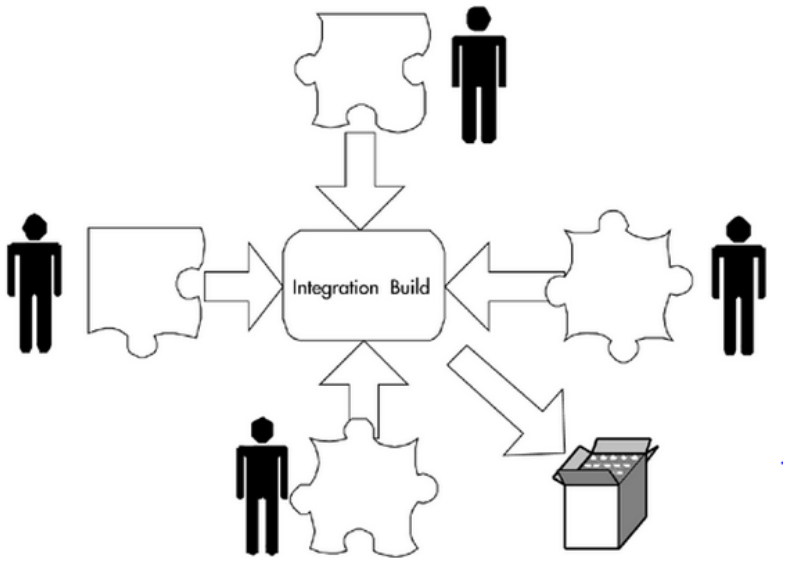

Figura 23. La integración centralizada ensambla las piezas [13]

La construcción debe automatizarse teniendo en cuenta:

- El tiempo que tome la construcción
- La velocidad con que se realicen los cambios.

Si el sistema toma mucho tiempo para construirse hay que decidirse por una construcción diaria.

Si la construcción no toma mucho tiempo, se puede considerar la construcción cada vez que se suban cambios al control de versiones.

Cada una de estas construcciones debe etiquetarse en el sistema de control de versiones.

Problemas no resueltos.

Aunque el sistema se construya de forma correcta podría no funcionar, para lo que se recomienda realizar una prueba de humo.

Si la versión generada va a liberarse a cliente hacer una prueba de regresión.

Implementación

La implementación más rudimentaria es asignar el trabajo de integración y compilación a una persona del equipo. Cada día o según lo que se haya determinado integrará y construirá el software.

Lo más recomendable es el uso de una herramienta. En el mercado hay varios servidores de integración continua que automatizan el proceso de construcción[4]. En ese

[4] Jenkins es un servidor de integración continua de código abierto compatible con muchos lenguajes de programación.

caso se puede implementar una rama de integración en el repositorio donde se hará la fusión de todos los cambios individuales.

4.2.9 Usar herramientas

Figura 24. Herramientas
[www.pixabay.com]

Los procesos de administración de la configuración son repetitivos por lo que en el mercado hay muchas herramientas que se pueden implementar. Servidores de integración continua, analizadores de código estático, despliegue automático, pruebas automatizadas forman parte de las opciones existentes.

Para empresas pequeñas se presentan opciones en la nube que minimizan los costos de implementar estas herramientas. Las características de algunas que se pueden usar se describen en adelante.

4.2.9.1 Despliegue automatizado

Es un complemento de la integración continua y requiere un alto grado de madurez del proceso.

En la medida que las integraciones generan pocos errores se pueden crear procedimientos automáticos que instalen en los ambientes de pruebas o pruebas cliente los componentes construidos.

En un mayor grado de madurez se puede automatizar este paso a producción. Sin embargo, los ambientes productivos suelen estar supeditados a las necesidades del servicio. Dada esta situación, lo recomendables es que la instalación en producción deba depender de la instrucción de un operario y de las debidas autorizaciones.

Un correcto despliegue automático o semiautomático debe cumplir:

1. Tener un mínimo de comandos, un clic o un comando de una línea.

2. Identificar que versión se va a instalar.
3. Generar un registro de la instalación.
4. Ofrecer la posibilidad de volver a la instalación anterior con un mínimo de esfuerzo.

La forma de implementar esto suele ser la construcción de un “script de actualización”. Este se encarga de verificar las dependencias, de instalar las nuevas versiones desde un repositorio oficial, verificar el estado de la base de datos, si la hay. Y realizar el registro de la nueva instalación.

Un buen script de instalación y actualización se logra con la práctica y con muchas pruebas, pero es algo que vale la pena por el tiempo que ahorra.

Un buen equipo de desarrollo debe aspirar a este estado.

En general las herramientas modernas de construcción de software ofrecen la elaboración de estos instaladores. Pero en el caso que trabajemos un software de vieja data es posible que debamos construirla manualmente.

4.2.9.2 Etiquetar versiones

Cada conjunto de componentes entregado para producción debe poder identificarse, para ello sirven las etiquetas de versión.

Se debe tener una forma estandarizada de etiquetar. La fórmula X.Y.Z, suele ser la más usada.

En este sistema el tercer digito corresponde a cambios pequeños, el segundo a cambios de base de datos y de interfaz de usuario y el primero a cambios del núcleo del software.

También se puede utilizar un cuarto digito para identificar el número de compilación que es un consecutivo simple que crece sin importar la versión.

Así una versión 1.2.3.35, es la compilación 35 del software y en esta se liberó la versión 1.2.3

Cuando se entrega una versión es muy recomendable guardar una copia del código fuente, esto con el fin de buscar posibles errores y de poder comparar futuras versiones.

La mayoría de sistemas de control de versiones cuenta con esta posibilidad, y se llaman Tags o etiquetas. Para rápida identificación se debe llamar a esta copia igual que el número de versión.

4.2.9.3 Los archivos de configuraciones del software

Para su correcto funcionamiento los programas requieren de otro software, bases de datos, archivos, repositorios, librerías, usuarios, contraseñas, web services, entre otros.

Si estos componentes están dispersados en muchas partes del software su administración se hace muy difícil, adicional a que es posible que se cometan errores si hay información duplicada.

Para evitar esto su administración debe estar centralizada en uno o varios archivos de configuración, el cual debe ser de pleno conocimiento de todo el equipo. Y estar en la base de conocimientos.

Los valores correspondientes a cada ambiente: desarrollo, pruebas, producción deben estar a cargo de la gestión del conocimiento. Y el acceso a esta información limitada a los niveles de autorización debidos.

No contar con esta información de forma centralizada puede llevar al equipo a cometer errores muy fáciles de corregir, pero fatales para la operación.

Estos archivos deben tener control de versiones y usar alguna herramienta para su administración.

4.2.10 Métricas de administración de la configuración

La implementación de cada uno de los patrones descritos crea la base para un marco de desarrollo ágil que es lo que se considera más adecuado para el mantenimiento del software.

Este es uno de los procesos que más medidas puede dar, por ejemplo:

- Cantidad de líneas de código
- Cantidad de desarrolladores
- Cantidad de compilaciones
- Tiempo entre integraciones
- Numero de errores de integración
- Puntos de función por periodo de tiempo
- Resultados de pruebas automatizadas (porcentaje de errores, cobertura de las pruebas, etc.)
- Cumplimiento de las reglas de programación

Dependiendo que estrategia se implemente se usará uno o varios de estos indicadores en el cuadro de mando integral. (Ver 4.9)

4.2.11 Resumen de administración de la configuración

La construcción, modificación, corrección y operación del software requiere una estructura tecnológica que lo respalde. Organizar estos recursos puede ser problemático para un equipo de trabajo.

Los patrones de diseño proponen una solución eficiente a la organización de los productos de trabajo del software. Y son fácilmente adaptables al marco de procesos que se use.

Un detalle de las políticas empresariales, de los recursos que se requieren para construir el software y un control de versiones del código fuente son la base para controlar con eficiencia los cambios del software. Esto en conjunto a la gestión de activos que se detalla en el capítulo 4.6.

Si esto se combina con integración y despliegue continuo. Pruebas automatizadas, y métricas del proceso se tiene lo necesario para la implementación de un marco ágil de desarrollo. Que se detalla en el próximo capítulo.

En el libro citado se describen más patrones de administración de la configuración, los descritos en el presente capítulo son los que se consideran mínimos, para una aceptable gestión de los cambios.

4.3 IMPLEMENTAR DESARROLLO ÁGIL

Figura 25.Pasos para Implementar desarrollo ágil
[Elaboración propia]

El siguiente paso para organizar el mantenimiento es establecer la forma en que se construirá el software, y lo más recomendable es implementar una metodología ágil, veamos de que se trata

Cambios pequeños, entregas frecuentes

En 2001 se reunieron 17 especialistas del desarrollo de software para firmar lo que llamaron el Manifiesto ágil. Ellos eran críticos de los modelos basados en procesos, y en su reunión decidieron llamar métodos ágiles a la alternativa que surgía a los métodos formales. Esa alternativa consiste en realizar entregas frecuentes de software útil a los usuarios.

Desde entonces han surgidos diferentes marcos de desarrollo ágil, siendo los más conocidos:

- Programación extrema (Xtreme programming)
- Scrum

Cada uno de estos marcos da una interpretación a la forma de implementar los valores del manifiesto ágil.

4.3.1 Valores del manifiesto ágil

Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:

*A los **individuos y su interacción**, por encima de los procesos y las herramientas.*
***El software que funciona**, por encima de la documentación exhaustiva.*
***La colaboración con el cliente**, por encima de la negociación contractual.*
***La respuesta al cambio**, por encima del seguimiento de un plan.*

Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda. [15]

Todos los marcos parten del hecho que las necesidades cambian, por lo tanto, cambian los requerimientos y en consecuencia el software.

La agilidad comprende ese cambio y en cierto modo lo alienta. Y para la implementación de estos cambios sugiere la entrega frecuente de incrementos del software, ello se expresa en los principios de la agilidad.

4.3.2 Principios del Manifiesto Ágil

Seguimos estos principios:

Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

El software funcionando es la medida principal de progreso.

Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.

La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia. [15]

Debido a los valores y principios de la agilidad, son los marcos de desarrollo ágil los más convenientes para la etapa de mantenimiento del software.

En estos marcos se da énfasis al desarrollo de software útil y se minimiza la documentación.

El capítulo sobre administración de la configuración es vital para implementar un marco ágil. Sin un control de versiones, repositorio de recursos o integración continua, la agilidad se puede ir por la borda.

El fin de la agilidad es consolidar un equipo que responde adecuadamente a los cambios en cualquier etapa del desarrollo. Agilidad es una forma de trabajo independiente del proceso de software, pero que está atado a él.

El fin no es solo responder a los cambios sino atarse a los principios y valores de la agilidad. Dar mayor prioridad al software útil, a las interacciones entre las personas y ser útil al cliente.

Esto se alinea con lo planteado de ser útil al modelo negocio.

Y esto se capitaliza con entregas frecuentes de software útil.

Ahora se exponen las generalidades de dos de los marcos más usados.

4.3.3 Elegir un marco ágil

Cualquier marco que se implemente debe incluir una estrategia de desarrollo incremental. Entregar incrementos de software, entendidos estos como ejecutables usables, en periodos cortos de tiempo. De modo que los incrementos vayan al ritmo de las solicitudes de cambio, o por lo menos lo más cerca posible de las expectativas del cliente.

Además de los incrementos, la conformación del equipo humano es vital. El equipo debe ser competente en su campo de acción, en la tecnología usada. Debe ser capaz de colaborar entre sí, debe poder resolver problemas, decidir la solución adecuada de estos problemas y debe tener confianza y respeto. Esto solo se logra con un gran liderazgo, lo cual suena contradictorio ya que se supone que los equipos ágiles se auto gestionan, así que el liderazgo debe surgir de cada miembro del equipo.

A continuación, se describen las generalidades de los dos métodos agiles más usados, Programación extrema y Scrum.

4.3.3.1 Programación extrema o XP

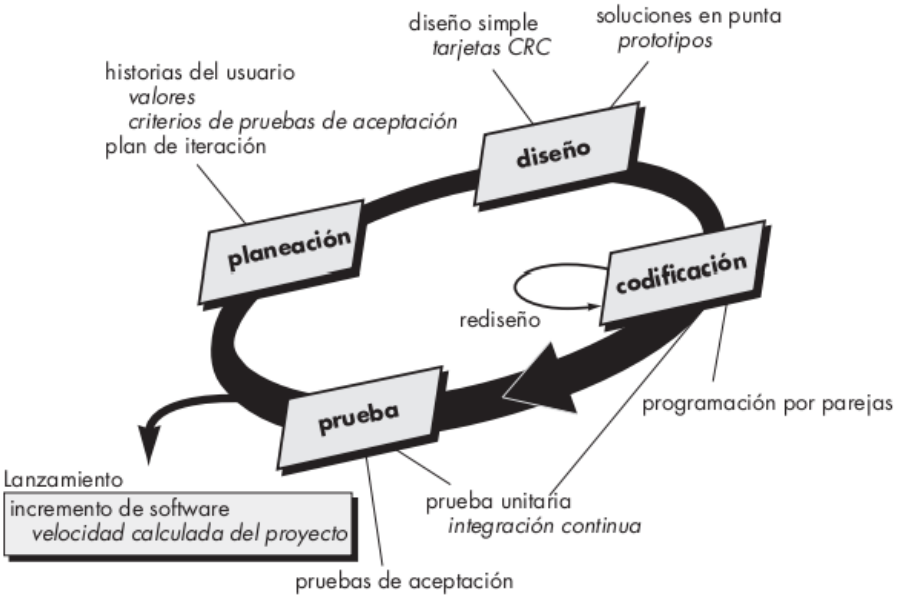

Figura 26.Flujo de trabajo programación extrema [16]

El proceso XP define cuatro actividades: planificación, diseño, codificación y pruebas.

Planificación inicia cuando el personal técnico escucha al cliente y escribe los requisitos en historias de usuario. El cliente asigna a estas historias un valor para mostrar la prioridad. El equipo técnico evalúa la historia y le da un costo medido en semanas de desarrollo. Si el costo es mayor a tres semanas, se pide al cliente que divida la historia y se le asigna valor y costo a estas sub-historias.

Los clientes y desarrolladores se reúnen para definir como agrupar las historias y acordar la próxima entrega. Una vez definido el cronograma el equipo debe ir midiendo la velocidad de desarrollo, como la cantidad de historias de usuario implementadas por incremento. Así el equipo se auto corrige para futuras planificaciones.

En **diseño** se sigue el principio mantenlo sencillo. Un diseño simple se prefiere a uno más complejo y no se realiza diseño adicional porque un desarrollador cree que se va a necesitar después.

Un concepto en XP es que el diseño ocurre tanto antes como después de la codificación. Se estimula el rediseño como forma de mejorar el diseño existente. También se propone el uso de tarjetas clase-responsabilidad-colaboración (CRC) [16]

En el paso de **codificación** se realizan una serie de pruebas unitarias que debe cumplir el desarrollo una vez esté realizado. Así el desarrollador está más capacitado para programar un código que debe pasar la prueba.

Se estimula la programación por parejas como forma de solucionar los errores en tiempo real.

En **pruebas** se ejecutan las pruebas unitarias que se han venido construyendo, lo que implica pruebas de regresión cada vez que se modifica el software. Se promueve la creación de pruebas automáticas para integración y validación.

Las pruebas de Cliente son definidas por este y se centran en las características de la solicitud hecha en las historias de usuario.

4.3.3.2 Scrum

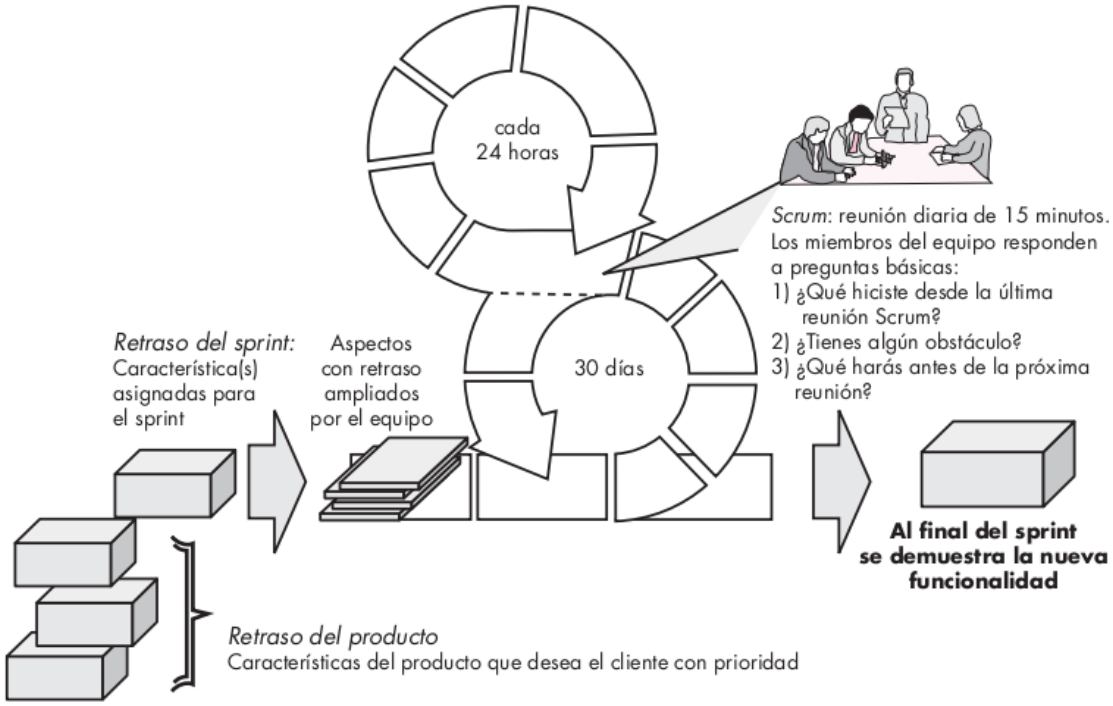

Figura 27. Flujo de trabajo Scrum
[16]

Scrum toma su nombre de cierta jugada de un partido de rugby. Sus principios son congruentes con el manifiesto ágil y tiene las siguientes actividades:

Requerimientos, análisis, diseño, evolución y entrega.

Cada incremento se construye en un periodo de tiempo fijo llamado Sprint, de entre dos a seis semanas.

El backlog, retraso o pila de producto es el lugar donde se acumulan las solicitudes de cliente en forma de historias de usuario. En cualquier momento se pueden agregar solicitudes.

Se realiza una reunión de planificación del sprint en la cual el equipo define que historias de usuario construirá en el sprint. Asignando a cada una un valor que expresa el tiempo que se debe dedicar a cada una.

Cada día se realiza una reunión de no más de quince minutos, donde el maestro scrum (scrum master) pregunta a cada uno:

¿Que hizo desde la última reunión?
¿Qué dificultades tiene/tuvo?
¿Qué va a hacer hasta la próxima reunión?

Con estas preguntas se espera que se muestren las dificultades que pueda tener el equipo y tomar los correctivos de forma oportuna.

Al final del sprint se entrega una versión del software que el cliente puede evaluar. Y se inicia un nuevo ciclo.

No necesariamente es una versión para poner en producción, pero si debe ser útil y completa.

4.3.4 Implementar el marco ágil elegido

Existen otros marcos de desarrollo ágil: Kanban, proceso unificado ágil, desarrollo impulsado por características, entre otros.

Un buen marco debe tener en cuenta los valores y principios del manifiesto ágil y sus actividades deben ser trazables. Saber quién hizo que, cuando y por qué es indispensable en cualquier marco adoptado.

También cumplir las prácticas de proceso. Si se tiene un marco de procesos como CMMI, ISO Spice u otro. El proceso ágil debe garantizar que se cumplen las prácticas. Es de anotar que estos procesos dan el que, pero no el cómo, por lo que no debe ser difícil mostrar las prácticas en el marco ágil adoptado.

La formación del equipo humano es muy importante. Un equipo con alta rotación que requiere continuos entrenamientos a personal nuevo puede afectar gravemente el rendimiento total. El proceso debe ser lo más simple que sea posible, y poder ser explicado en un pequeño manual o una capacitación muy corta. Y claro todo esto debe formar parte de la base de conocimientos.

La implementación consiste en la documentación de los procesos, la socialización de los mismos y la preparación del material de capacitación para nuevo personal. De allí en adelante el proceso debe auto corregirse en la medida que se midan resultados y se tomen correctivos.

4.3.5 Seguimiento y control

Los marcos ágiles están de moda. Muchas industrias y grupos de organizaciones dicen ser ágiles. Y los marcos son bastante abiertos para incluir cualquier cosa. Por lo que es necesario medir corregir y volver a medir para saber qué tan efectivo es el marco implementado.

Medidas de calidad del producto y productividad darán las bases para ajustar la forma de trabajar. Esto es indispensable

4.3.6 Métricas de desarrollo

En desarrollo y teniendo en cuenta el marco adoptado se pueden obtener múltiples medidas:

- Tamaño de la pila del producto; es decir de las solicitudes no atendidas.
- Velocidad del equipo. Para resolver un error o una solicitud.
- Defectos promedio por entrega.
- Reprocesos dentro del ciclo.
- Numero de historias de usuario y/o puntos de función construidas por modulo/desarrollador/día.
- Numero de defectos no resueltos.

4.3.7 Resumen de desarrollo ágil

El desarrollo ágil da prioridad al software que funciona sobre la documentación. Promueve la entrega frecuente y la comunicación efectiva y constante con el usuario.

Implementar un marco ágil requiere personal multidisciplinario y eficiente que se comunique de forma permanente.

Un marco ágil puede llevar a que se optimicen los recursos sin descuidar la calidad, siempre y cuando se tenga un buen aseguramiento de la calidad, que se trata en el próximo capítulo.

4.4 IMPLEMENTAR ASEGURAMIENTO DE LA CALIDAD

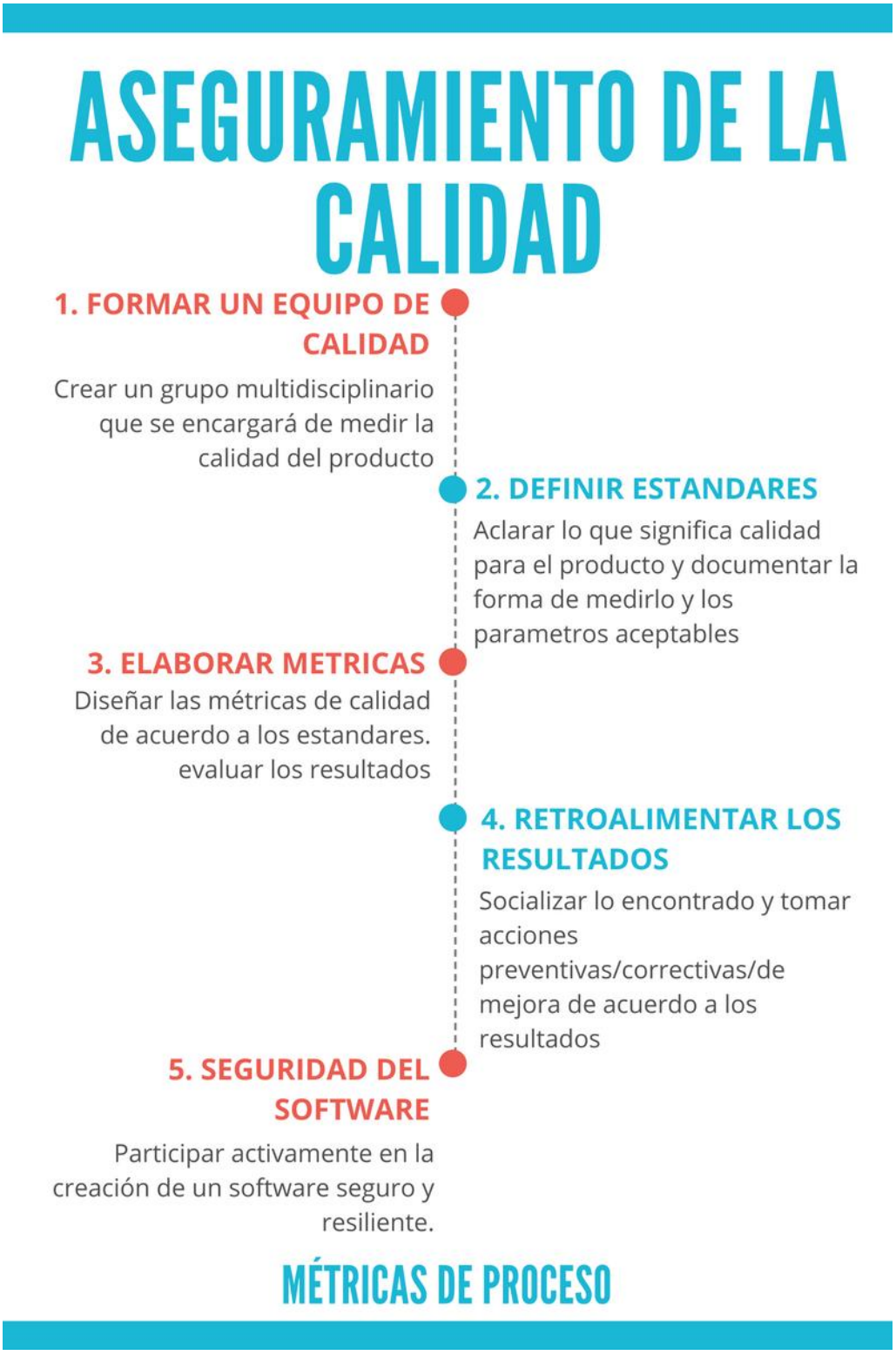

Figura 28. Pasos para el aseguramiento de la calidad del producto [Elaboración propia]

El aseguramiento de la calidad será el encargado de recolectar los datos para construir los indicadores y analizarlos. Es el proceso encargado de llevar las cifras, incluyendo las que se usen en el cuadro de mando integral.

La calidad en el software tiene muchas interpretaciones. En general se refiere a lograr un producto que satisfaga las solicitudes del cliente y que se entregue dentro del plazo y el presupuesto. Esto no es fácil de medir ya que muchos conceptos son subjetivos.

Un adecuado proceso de aseguramiento de la calidad permite que el software satisfaga las necesidades del cliente hasta donde es técnicamente posible, que tenga un desempeño confiable y agrega valor a todos los que lo utilizan.

Para lograr estos objetivos, se propone que se realicen tres actividades que se pueden enmarcar en cualesquiera que sean los procesos de desarrollo de software:

1. Formar un equipo de calidad.
2. Elaborar métricas.
2. Adoptar técnicas de desarrollo seguro.

Todo esto englobado en la administración de la operación que incluye el desarrollo y la ejecución del software en el entorno productivo.

Se sabe que entre más recursos se dediquen a calidad, mejor será el producto. Pero no se puede caer en la tentación de tener un proceso sin final para calidad. Cientos de compañías alrededor del mundo sacan al mercado productos que son “suficientemente buenos”, pero no perfectos.

La pregunta es ¿cuánto es suficientemente bueno? La respuesta es: depende del tipo de producto. En un software de aviación suficientemente bueno es mucho más complejo que en un software de juegos.

Así que la cantidad de calidad que se requiere depende de las mediciones que se realicen. Un indicador irá diciendo en el tiempo cuanta actividad de calidad se requiere. Y ello lo define el grupo de calidad.

4.4.1 Formar un equipo de calidad

El equipo de calidad se forma con personas del equipo con diferentes especialidades, algunas se dedicarán de tiempo completo a actividades de aseguramiento de la calidad y otras estarán de forma parcial, asesorando y afinando actividades.

Debe tener autoridad para aceptar o rechazar los cambios hechos por el equipo de desarrollo.

Es responsable de las siguientes actividades [17]

4.4.1.1 Definición que es calidad

ISO, IEEE, y otras organizaciones definen una serie de estándares del software. Estos van desde la confiabilidad a la usabilidad. El grupo de Calidad debe definir en el marco del negocio cuales son los estándares por usar y la forma de medirlos.

En algunos casos los estándares están definidos contractualmente por el cliente.

4.4.1.2 Revisiones y auditorías

Las revisiones son una actividad, hecha por profesionales pares para la búsqueda de errores. Entre más tarde en el ciclo de vida se detecte un error, más costoso es corregirlo. Las revisiones buscan detectar de forma temprana los errores. Se recomienda que se hagan desde los documentos de análisis.

Las auditorías son actividades de revisión, cuyo objeto es garantizar que se sigan los lineamientos de los procesos y recomendar las acciones preventivas y correctivas que sean necesarias.

4.4.1.3 Pruebas de software

Las pruebas deben detectar errores. La función del equipo de calidad es que las pruebas se planeen de forma adecuada y se ejecuten de forma eficiente.

Las técnicas de pruebas aplicadas dependen de la criticidad del software y de la tecnología usada.

4.4.1.4 Análisis de los errores

La única forma de saber que se está haciendo mal es medirlo. El grupo de calidad debe reunir y analizar los datos acerca de los errores, entender mejor cómo se cometen, y definir las actividades para eliminarlos.

4.4.1.5 Administración del cambio

El cambio irrumpe en cualquier proceso de software. El grupo de calidad debe verificar que se instituyan prácticas adecuadas de administración del cambio. Una mala administración del cambio lleva a mala calidad. Esto está muy atado al proceso de administración de la configuración.

4.4.1.6 Administrar la educación

El grupo de calidad lleva el liderazgo en los programas educativos que se realicen en el equipo de software.

Esto debe ser un componente crucial de los procesos de mejora continua.

4.4.1.7 Administrar la seguridad

Los delitos cibernéticos han venido en aumento. Se deben definir políticas para el desarrollo de software seguro. Así como para definir la seguridad permitida. Identificar las vulnerabilidades y asegurarse que se implementen los correctivos.

Además de ello debe velar porque se cumpla la normativa vigente acerca de protección de datos. y la seguridad del software[5]

4.4.1.8 Administración del riesgo

Cada riesgo debe ser identificado y tener un plan de contingencia documentado en caso de que se materialice.

[5] Por su importancia 4.4.5 detalla más este punto

El fin último del grupo de calidad es asegurar las actividades del desarrollo software y ver que se realicen y que esto sea con la mayor calidad posible.

4.4.2 Definir estándares

Se dice que la calidad es cómo el sexo. Todo el mundo sabe que es, pero no lo puede explicar. Para no caer en un limbo, el grupo de calidad debe definir qué es lo que se considera calidad. Cómo va a medirse y los mínimos esperados de esa medida.

Hay medidas que pueden tomarse directamente, como el número de fallos. Y otras tienen mediciones indirectas como la usabilidad del software. Por lo que la tarea de definir estándares a medir no es fácil. Existe mucha literatura al respecto.

4.4.3 Elaborar métricas

La calidad del producto es responsabilidad de dos grupos que trabajan en conjunto: los ingenieros de software que hacen el trabajo técnico y el grupo de calidad.

El primero se dedica a las labores técnicas como escribir el código, asegurar el cumplimiento de los estándares y almacenar los datos que se usarán para las mediciones.

El segundo se dedica a planear, supervisar, analizar y hacer reportes acerca de los números encontrados.

Ambos grupos abordan la calidad aplicando métodos sólidos, realizando revisiones técnicas y haciendo pruebas bien planeadas. Para ello deben realizar las siguientes tareas en el proceso de construcción de software:

- Elaborar el plan de calidad del software, este define lo que se va a considerar como calidad y la forma de medirlo
- Participar en la descripción del software. Esta descripción debe estar dentro de las políticas del negocio y cumplir la política organizacional
- Revisar las actividades de ingeniería del software. Con el fin de verificar el cumplimiento de los procesos definidos.
- Auditar los productos de trabajo que se designen para verificar. Se verifica el cumplimiento, que se hayan realizado las correcciones
- Documenta las desviaciones del proceso y asegura que se manejen según el proceso documentado.
- Registra la falta de cumplimiento y la notifica a la dirección. Se debe reportar a quien tenga la autoridad para que esté informado y tome las acciones correctivas o de negociación pertinentes.

Para lograr esto se deben elaborar métricas.

4.4.3.1 Que deben medir las métricas

Las acciones anteriores se definen para alcanzar un conjunto de metas medibles de calidad:

- Calidad de los requerimientos

- Calidad del diseño
- Calidad del código
- Eficacia del control de calidad

La calidad de los requerimientos debe medir la completitud, corrección y consistencia de los requisitos. Estos tendrán una gran influencia en los productos que le sigan.

La calidad del diseño debe ser evaluada para verificar si cumple los requerimientos y tiene una buena calidad de diseño, esto es especialmente importante en programación extrema que propone un diseño incremental.

La calidad del código evalúa el cumplimiento de estándares, y la facilidad para dar mantenimiento.

La eficacia del control de calidad evalúa la asignación de recursos a calidad ya que estos son siempre limitados.

La siguiente figura muestra algunas de las métricas que pueden ser evaluadas.

Meta	Atributo	Métrica
Calidad de los requerimientos	Ambigüedad	Número de modificadores ambiguos (por ejemplo, muchos, grande, amigable, etc.)
	Completitud	Número de TBA y TBD
	Comprensibilidad	Número de secciones y subsecciones
	Volatilidad	Número de cambios por requerimiento
		Tiempo (por actividad) cuando se solicita un cambio
	Trazabilidad	Número de requerimientos no trazables hasta el diseño o código
	Claridad del modelo	Número de modelos UML
		Número de páginas descriptivas por modelo
		Número de errores de UML
Calidad del diseño	Integridad arquitectónica	Existencia del modelo arquitectónico
	Completitud de componentes	Número de componentes que se siguen hasta el modelo arquitectónico
		Complejidad del diseño del procedimiento
	Complejidad de la interfaz	Número promedio de pasos para llegar a una función o contenido normal
		Distribución apropiada
	Patrones	Número de patrones utilizados
Calidad del código	Complejidad	Complejidad ciclomática
	Facilidad de mantenimiento	Factores de diseño (capítulo 8)
	Comprensibilidad	Porcentaje de comentarios internos
		Convenciones variables de nomenclatura
	Reusabilidad	Porcentaje de componentes reutilizados
	Documentación	Índice de legibilidad
Eficacia del control de calidad	Asignación de recursos	Porcentaje de personal por hora y por actividad
	Tasa de finalización	Tiempo de terminación real versus lo planeado
	Eficacia de la revisión	Ver medición de la revisión (capítulo 14)
	Eficacia de las pruebas	Número de errores de importancia crítica encontrados
		Esfuerzo requerido para corregir un error
		Origen del error

Figura 29.Algunas métricas de calidad del software [16]

4.4.3.2 Qué hacer con las métricas

Cuando se mide se sabe en que se está fallando. El siguiente paso es aplicar los correctivos para lograr mejor calidad.

Una de las formas más eficiente de realizar esta tarea es usar el principio de Pareto: El 80% de los defectos se debe al 20% de las causas.

La manera de aplicar Pareto es:

1. Se clasifican los errores encontrados de forma numérica.
2. Se rastrean los errores hasta su origen, como puede ser mala especificación, error de diseño, mala comunicación con el cliente, etc.
3. Se identifica el 20% de las causas y se aplican correctivos.
4. Se realiza de nuevo la medición, si fue efectiva hay un nuevo 20% de causas, si no hay un nuevo 20%, se deben cambiar los correctivos
5. Repetir el ciclo.

Con este sistema se pueden reducir muchos errores y se tiene la ventaja de tener un proceso de mejora continua.

4.4.4 Retroalimentar los resultados

Los resultados encontrados deben usarse, de lo contrario no aportaran a la calidad final del producto. De acuerdo a lo encontrado deben realizarse actividades cómo:

1. Corregir los procesos de desarrollo para reducir los errores.
2. Informar acerca de la productividad.
3. Mejorar el desempeño/seguridad/ de los aplicativos.
4. Producir conocimiento.

4.4.5 La seguridad del software

Durante años la seguridad informática ha sido tema de expertos en esa área. Personal especialmente capacitado investiga las vulnerabilidades y encuentra la forma en que se produjeron o se pueden producir los delitos informáticos. Pero estas vulnerabilidades tienen su principal fuente en el código que escriben los ingenieros de software.

Por esta razón es indispensable que el equipo de desarrollo tenga en su proceso una forma de no introducir errores y vulnerabilidades al software que construye.

Para minimizar la existencia de errores y vulnerabilidades se propone que dentro del proceso de construcción del software se implementen prácticas que conduzcan a este propósito.

Lo que se propone se basa en lo que postula Merkow, [18] que propone que se debe fabricar software seguro y resiliente. Entendiendo seguro como software libre de vulnerabilidades conocidas, y resiliente con que tenga gran capacidad para recuperarse de los incidentes de seguridad.

4.4.5.1 Errores y vulnerabilidades

Hoy, el software está en todas en todas partes. Desde complejos dispositivos de navegación en nuestro auto, hasta el dispositivo móvil que cabe en nuestra mano. Todo este software influye de alguna forma en la civilización en que vivimos. Y todo tiene algo en común: fue escrito por personas.

Las escuelas de formación de programadores, ingenieros de software, de computación y otros profesionales de las tecnologías; no han tomado en cuenta la necesidad de la seguridad del software. Sólo en los últimos años y ante la creciente incidencia de delitos

informáticos los profesionales de la programación han vuelto su rostro a los problemas de seguridad.

Según los análisis, se presentan dos tipos de problemas de seguridad: Errores y vulnerabilidades.

Los errores son segmentos de código que hacen que el software no funcione como se espera. Estos errores pueden ser introducidos de forma intencional o por error humano.

Las vulnerabilidades son defectos del software que permiten que un atacante tome control parcial o total del software.

Tanto los errores como las vulnerabilidades se pueden minimizar con el uso de requisitos no funcionales. En general los requisitos solo se refieren a la forma como debe trabajar el programa, y algunos requisitos no funcionales tratan de la carga o la capacidad de responder a la cantidad de usuarios.

Unos buenos requisitos no funcionales cubren la capacidad de recuperación ante los incidentes y la seguridad, entendida esta como la posibilidad de detectar y rechazar un ataque.

Estos requisitos son dictados por el personal de apoyo, analistas, normas gubernamentales y de la empresa. Es importante hacer entender al cliente de la importancia de estos requisitos para que esté dispuesto a conceder el tiempo y otros recursos necesarios para su desarrollo.

El levantamiento de requisitos no funcionales se puede realizar de la siguiente forma:

- Comenzar con una base de conocimientos de seguridad para los sistemas que son similares en el área de desarrollo.
- Determinar si un atacante puede tener motivos para intentar un ataque al negocio. Analizar las posibles vulnerabilidades.
- Hacer una lluvia de ideas sobre los casos de uso, de lo que podría salir mal. Por ejemplo, analizar el uso válido y luego los pasos maliciosos

Elaborar estos requisitos puede ser un trabajo extenuante, la gran ventaja es que son reutilizables para otros proyectos.

4.4.5.2 Prácticas que se deben introducir en el proceso de desarrollo

Son fácilmente adaptables a cualquier tipo de metodología que se use:

1. En los estándares de calidad introducir los requisitos de seguridad y resiliencia.
2. Dentro de los requisitos, documentar los requisitos no funcionales.
1. El proceso de pruebas debe incluir la prueba de los requisitos no funcionales.

Los requisitos no funcionales son difíciles de probar. Y en general tienen una evaluación subjetiva. La forma de probarlos es con ataques directos al software y a la respuesta que dé a cada ataque se le da un grado de resiliencia. Este tipo de pruebas es hecho por expertos en seguridad.

Las revisiones técnicas del código fuente ayudan a encontrar el código mal intencionado introducido de forma deliberada, y también ayuda a encontrar vulnerabilidades introducidas por error humano.

Aunque se entiende la dificultad que esto aporta es indispensable que se implementen estos requisitos para producir software de calidad. Sobre todo, cuando se trata de software de vieja data que puede haber sido construido con muchos problemas de seguridad.

Un ejemplo de requisitos no funcionales se da en la Tabla 1, estos se basan la familia de requisitos propuesta por Merkow [18] y son para un software de control de préstamos de libros de una biblioteca universitaria

Tabla 1. Requisitos no funcionales de control de préstamos de libros en biblioteca universitaria.

Titulo	**Descripción**
Disponibilidad	Debe estar disponible en el horario de la biblioteca 7:am a 9:00 pm de lunes a viernes
Capacidad	Debe soportar todos los equipos en que esté instalado: 20 equipos inicialmente.
Eficiencia	Soportar al menos 20 transacciones simultaneas.
Extensibilidad	Debe poder ampliarse fácilmente a mas equipos donde este instalada
Interoperabilidad	No se contempla inicialmente que deba operar con otros sistemas
Gestionabilidad	La creación de usuarios, cambios de contraseñas, carga de inventario debe poder ser hecho por los usuarios administradores
Mantenibilidad	El código fuente debe ser fácilmente identificable las funcionalidades que realiza, debe estar documentado de modo que se fácil corregirlo. Debe crearse documentación funcional y técnica del software
Desempeño	Debe soportar que todos los usuarios estén conectados y operando el software, aproximadamente 13 usuarios estudiantes y 7 usuarios bibliotecarios.
Privacidad	Los datos personales se usarán de acuerdo a la normatividad vigente de uso de información privada. Las estadísticas de préstamos de libros no deben violar la privacidad de los usuarios.
Recuperabilidad	El sistema debe recuperarse de un incidente en máximo tres horas en horario de servicio de la biblioteca.
Confiabilidad	El software debe operar correctamente en todo el tiempo del servicio. No debe deteriorarse por el uso, por la conexión de más usuarios o un corte de energía
Soportabilidad /Servicio	Debe crearse una mesa de ayuda para el soporte del sistema en el horario de operación de la biblioteca

[Elaboración propia]

4.4.5.3 Algunos requisitos de seguridad que se deben documentar y probar

Los requisitos de seguridad cubren por un lado la seguridad perimetral, es decir, cubrir la posibilidad de recibir un ataque de forma física y también la seguridad tecnológica del software. Ambos requisitos deben ser cubiertos.

Tsipenyuck [19] propone una taxonomía para la documentación de requisitos de seguridad, estos tipos de requisitos pueden ser fácilmente adaptables a cada necesidad.

Un ejemplo del uso de esta taxonomía es:

Requisitos de identificación
La identidad del usuario debe ser fácilmente rastreable. Se debe poder saber la identidad completa del usuario que hace cada operación

Requisitos de autenticación
Todo usuario del software debe autentificarse. Esta identificación debe hacerse en el mayor nivel que sea necesario, por ejemplo, debe hacerse con algo que el usuario sabe: un usuario y contraseña y algo que tiene un carné de identificación y/o identificación biométrica.

Requisitos de autorización
Los usuarios pueden tener diferentes niveles de autorización de uso del software.

Requisitos de detección de intrusiones
Se debe limitar el número de veces que se puede introducir el usuarios y contraseña.
También se debe limitar el uso de:

1. Sql injection.
2. Buffer overflow.
3. Formatos de cadena de caracteres.

Se considera que para evitar esto se debe tener en cuenta:

No aceptar comandos SQL.
No aceptar caracteres especiales.
Limitar el número de caracteres.

Y agregar los requisitos que el diseño encuentre que deben agregarse. (estos son ejemplos de vulnerabilidades)

Requisitos de repudio
Los usuarios no podrán negar el uso del software

Requisitos de privacidad
Los datos personales de los usuarios deben ser tratados de acuerdo a la legislación vigente. No podrán ser usados para fines diferentes al uso normal del aplicativo.

Requisitos de auditoría
Las contraseñas de los usuarios deben estar encriptadas en la base de datos.
Todos los usos, cambios de contraseña y otras operaciones deben poder ser rastreadas hasta sus usuarios con fecha, hora e ip del equipo donde se originó la transacción.

Seguridad (física)
El acceso al centro de datos debe ser restringido solo a personal autorizado, y debe conservarse bitácora física del ingreso y salida de personal a este centro.

Requisitos de Supervivencia y recuperación ante el desastre.

Debe crearse un plan de recuperación ante el desastre. Debe incluir políticas de respaldo de la información. Instructivos para la reinstalación del sistema y protocolos de autorización para recuperar el sistema.

Requisitos de seguridad del mantenimiento del sistema.
El mantenimiento del sistema debe realizarse a partir de requisitos generados por la autoridad de la biblioteca.

Los nuevos requisitos aprobados para su desarrollo deben tener una sección de seguridad, y de requisitos no funcionales.

4.4.6 Resumen de aseguramiento de la calidad

La calidad del software debe ser algo medible y cuantificable. Se recomienda la creación de un grupo de calidad que tenga independencia para evaluar esa calidad con métodos cuantitativos.

Las actividades del grupo de calidad deben definirse, tomando en cuenta lo que se expone en esta memoria, complementándolos o disminuyéndolo según lo necesario. Como actividad indispensable se considera que debe ser el grupo encargado de recolectar los números para los indicadores del cuadro de mando y también los de los indicadores de procesos y los de la calidad del software.

Desarrollar software seguro y resiliente, es una de las asignaturas pendientes de los equipos de construcción de software. Es una tarea indispensable implementar prácticas de desarrollo seguro. La introducción y prueba de requisitos no funcionales ayudará mucho en esta tarea.

Las revisiones pares, auditorias y pruebas de software. Todo medido con métricas harán que la construcción del software sea cada vez de mejor calidad.

Ahora se expone otro proceso indispensable para el gobierno del software de mantenimiento: La gestión del cambio.

4.5 PROCESO DE GESTIÓN DE CAMBIOS DEL SOFTWARE

Figura 30. Pasos para la gestión de cambios
[Elaboración propia]

El esquema expuesto en 4.2 permite tener control de los archivos en un sistema robusto de administración de configuración. Sin embargo, se requiere definir las políticas de cómo se harán los cambios, y la forma cómo se promoverán a ambientes productivos. Esto es lo que se tratará en este punto.

Si se adopta una metodología ágil, se tendrán una serie de incrementos programados que se esperan cumplir. Pero es inevitable que existan cambios urgentes, debidos, por ejemplo, a cambios de la norma o de las políticas empresariales. O que, haya que liberar parches para solucionar errores o defectos del software.

Los cambios urgentes y los parches son un dolor de cabeza para los equipos de desarrollo, por lo que lo deseable es que salgan en conjunto con los incrementos programados. Si esto no es posible se debe tener un proceso definido, controlado y repetible para liberarlos. Este proceso debe cumplir con:

- Ser probado en la última versión liberada.
- Ser integrado y probado en las siguientes versiones
- Poder ser corregido en caso de nuevos errores o cambios urgentes.

4.5.1 Definir un proceso de control de cambios

Pressman [16] propone un proceso genérico de cambios que se muestra en la figura:

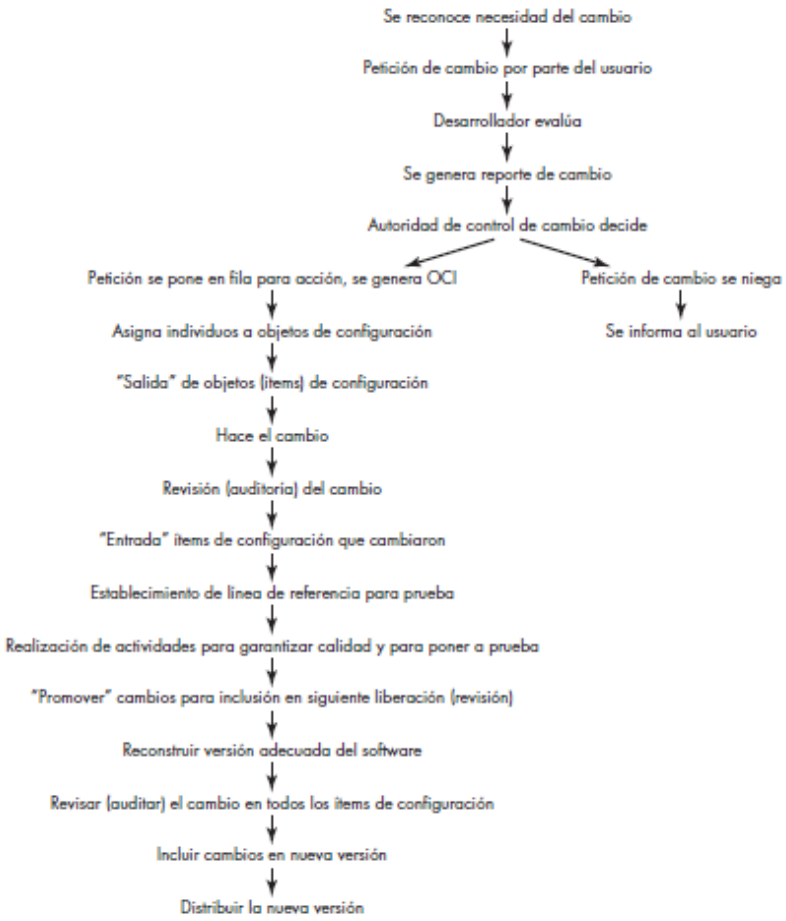

Figura 31. Proceso genérico de control de cambios [16]

Este proceso puede ser muy pesado para un software en mantenimiento, por lo que debe adaptarse a cada necesidad. A continuación, se detallan pasos que se consideran indispensables en cualquier proceso que se adapte.

4.5.2 Clasificar los cambios

Las solicitudes de cambios deben documentarse, organizarse y calendarizarse. Una posible clasificación de los cambios es:

Programados son aquellos que corresponden a una programación de liberaciones. De acuerdo con lo expuesto en 4.3, deben ser cambios pequeños, manejables en un periodo de desarrollo. En el calendario programado se define la prioridad.

Parches. Son las liberaciones hechas para corregir errores. Si el error es bloqueante deben liberarse una vez construidos. Si no es bloqueante puede ir a la fila de cambios programados. La fuente de solicitudes en es el soporte.

Urgentes. Son cambios que deben hacerse en el menor tiempo posible. Por ejemplo, para cumplir cambios de las leyes, normas, sentencias judiciales. Son solicitudes directas del cliente

Los cambios urgentes y los parches son los que se puede complicar el manejo. La administración de la configuración debe proporcionar el espacio de trabajo en el repositorio para hacer este tipo de cambios. La forma más sencilla de manejarlos es crear una línea de desarrollo a partir de lo que esté instalado en producción, que debe ser la línea principal, y allí, hacer los cambios. Una vez estén probados, liberarlos e integrar el cambio en la línea de los cambios programados y en la línea principal.

Evaluación del impacto
Todo cambio debe medir el impacto en el software en general y en el proceso, esta evaluación debe hacerse en conjunto entre personal técnico y administrativo. Y debe responder preguntas como:

¿El cambio solicitado afecta otros cambios que están en proceso?
Podría por ejemplo solicitar un cambio a un módulo o funcionalidad que está en proceso de construcción de otro cambio. Se debe evaluar el impacto de integrar los dos cambios, y podría negociarse la entrega de ambos cambios a la vez. O de detener y retrasar el cambio que está en proceso. Incluso suspender el cambio en proceso.

¿Qué tanto afecta el cambio?
Agregar un campo a una pantalla parece un cambio simple. Pero afecta el modelo de datos por lo que las pruebas pueden ser muy costosas en recursos. Un cambio como este es quizás candidato a ser un cambio programado. No deben subestimarse una solicitud de cambio por pequeña que parezca.

¿Qué sucede en el negocio si no se realiza el cambio?
Una solicitud puede ser clasificada cómo urgente por quien solicita. Esto puede ser porque sabe que los cambios urgentes se atienden de inmediato. Pero es una ventana abierta para que entren cambios de todo tipo.

Por ello deben responderse la pregunta a que pasa si no se resuelve. Todos los programas tienen errores conocidos y sobreviven con ellos porque resolverlos no aporta nada al negocio.

¿Cómo afecta el cronograma de entregas?
Los recursos de desarrollo son limitados y poner una solicitud nueva atrasa otra que tal vez es más importante. Los responsables del gobierno del proyecto y el cliente deben tener presente que un cambio urgente atrasará otras entregas.

Algunos administradores tienen la política de “el cronograma no se mueve”. Si es una práctica habitual, hará que el personal rote, lo que también atrasa el cronograma.

¿Afecta la continuidad del negocio?
Si poner en producción el cambio requiere que todos estén fuera del sistema, o se deben adquirir licencias, o software adicional. El impacto en el negocio puede ser alto.

El equipo de calidad debe garantizar que este parche o cambio urgente es probado en la siguiente versión entregada.

4.5.3 Resolver la solicitud

El cambio una vez calendarizado debe enviarse al personal adecuado para su solución, que puede ser un nuevo desarrollo. La liberación de un parche o de un procedimiento para resolver un error conocido.

Si es responsabilidad de desarrollo, la administración de la configuración es de vital importancia para evitar que el error se repita y que sea integrado en próximos desarrollos.

4.5.4 Control de calidad

Todo cambio debe tener control de calidad por pequeño que sea. Esto no es negociable.

Los casos de prueba deben ser lo más completos posible.

4.5.5 Gestor de cambios

El gestor de cambio es quien define el cómo se realizará el cambio. Es un rol que debe involucrarse con el desarrollo para saber cuánto tiempo tomará construir y probar el cambio. Y con operaciones para saber cómo afecta la continuidad del negocio.

Es quien decide la hora del cambio, si se debe reversar o continuar una vez hecho.

Debe informar a todos los involucrados del cambio y verificar que se haya cumplido el proceso de cambios. Llenar la documentación correspondiente, llevar los indicadores. Es un rol entre administrativo y técnico.

4.5.6 Métricas de gestión de cambios

Algunas medidas que se pueden tomar de este proceso son:

Porcentaje de cambios por tipo contra el total.
Días desde el último cambio.
Cambios pendientes por instalar.
Cambios por mes/año.

4.5.7 Resumen de gestión de cambios

La gestión de cambios minimiza el impacto en el calendario de liberaciones y la operación del software.

Es el proceso que más injerencia tiene para administrar los recursos humanos con que cuente el área. Ya que siempre que se introduzca una nueva solicitud se afectan estos recursos.

Clasificar los cambios, resolverlos, aplicarles control de calidad e implantarlos son las actividades que harán que este proceso tenga éxito. Y para que funcione se necesita una infraestructura que debe ser administrada. La gestión de activos es el próximo tema a tratar

4.6 PROCESO DE GESTIÓN DE ACTIVOS

Figura 32. Pasos para la gestión de activos
[Elaboración propia]

El software consiste en un conjunto de programas de computadora que se ejecutan y hacen algo útil.

Erwin Daniel Suarez Rodriguez. El gobierno del software en fase de mantenimiento

Para que esto sea posible se requiere de **hardware**: servidores, redes, instalaciones eléctricas…; que alojan el software. **Software**: Sistemas operativos, motores de bases de datos, programas comerciales, software de terceros…; que permiten la ejecución. Y también requieren de licencias, certificados de seguridad, e información de conexiones a bases de datos, usuarios de red, cortafuegos, puertos de salida y entrada, servicios web. Que hacen que el software se conecte correctamente. Toda esta información debe ser administrada y los cambios controlados. Un cambio no autorizado o no rastreable puede poner en peligro la operación de los programas y no es fácil encontrar la causa.

Una correcta administración permite volver a la versión estable, lo que garantiza la continuidad de la operación.

Y lo mas importante desde el punto de vista de esta memoria es que estos recursos cuestan dinero, lo que es de gran interés para la administración y es posible que se incluyan indicadores en el cuadro de mando integral.

Marcos para la organización
ITIL, Biblioteca de infraestructura de tecnologías de la información[6]. Da un marco de procedimientos para la administración de servicios de información. También el estándar ISO/IEC 20000. Cobit 5[7], Objetivos de control para la información y tecnologías relacionadas 5. Dan sus versiones de marcos de procesos para administración de servicios de información.

Los tres tienen en común que se trata de entregar servicios lo cual tiene sentido, ya que la operación del software es un servicio de tecnología.

Cualquier marco que se adopte, bien sea de los enumerados u otro. En general toma años su implementación. Ya que los marcos son extensos y su documentación, socialización y ejecución requiere un alto grado de madurez corporativa.

También se debe tomar en cuenta que para equipos de tecnología pequeños y/o pequeñas y medianas empresas, implementar un marco de procesos tiene un costo que la organización tal vez no puede asumir.

Por ello, se expone acerca de lo mínimo que debe existir para tener una aceptable gestión de los activos.

4.6.1 Crear una CMDB

La CMDB, base de datos de administración de la configuración[8], es la fuente de la información de los activos. En ella se especifican los activos y sus atributos. Los activos se les llama elementos de configuración. Cada elemento tiene unas propiedades y relaciones con otros elementos.

Entre las primeras cosas que debe organizar un grupo de tecnología, es una CMDB y el proceso de cambios para los elementos de configuración.

Dado que se tienen tres tipos de elementos: hardware, software e intangibles. ITIL recomienda una base de datos federada; es decir; tres bases de datos que den reportes. Esto dependerá del software que se use para la implementación.

[6] ITIL en inglés: Information technology infrastructure library
[7] COBIT, en inglés: Control Objectives for Information and related Technology
[8] CMDB en inglés: Configuration manager data base

No todo lo que forme parte del área de tecnología es susceptible de ser registrado, por ejemplo, sillas, muebles, papelería, no deben registrarse en la CMDB. Si no, lo que es vital para la operación.

La CMDB no es un inventario.

4.6.2 Definir elementos de configuración.

Un elemento de configuración es una pieza susceptible de ser cambiada, y puede ser software, hardware o un intangible.

Para que un elemento de configuración pueda considerarse como tal debe cumplir las siguientes condiciones:

1. Es una pieza que puede ser identificada.
2. Puede cambiar con el tiempo.
3. Es necesaria para la operación del software.

Con estas condiciones son elementos de configuración: servidores, procesadores, memoria física, sistema operativo, cortafuegos, antivirus, ejecutables del software, conexiones a la red… Y todo lo que se requiera para mantener la operación.

No son elementos de configuración, los manuales de usuario, los planes de proyecto, facturas de compra…

El primer paso para la implementación de una CMDB es un inventario de los elementos de configuración.

4.6.3 Definir atributos de los elementos de configuración

Los atributos de los elementos de configuración son el conjunto de información que permiten su identificación. Cada elemento es único debido a estos atributos.

Cada producto para el manejo de una CMDB tiene su propia estructura. Algunos atributos que puede contener son:

Identificador: nombre del elemento de configuración

Versión: los elementos de software tienen una versión proporcionada por el fabricante.

Modelo: aplica en general a elementos físicos

Configuraciones: conexiones a bases de datos, puertos de salida, ip, url, número de serie y toda la información configurable. Se debe guardar la información precisa para casos de restauración de la información

Criticidad: una escala que puede darse de 0 a 3, siendo cero que no es necesaria para el negocio y 3 que es indispensable para la operación.

Fecha de caducidad: Una licencia se puede vencer. Un certificado de seguridad.

Instalador: nombre y ubicación del instalador en la biblioteca de configuración

Otra información que se considere puede ser almacenada, como fabricante, número de teléfono, directorio que debe almacenar, instrucciones para su ejecución.

4.6.4 Relaciones entre los elementos de configuración

Cada elemento puede o no relacionarse con otro, y depender de otro elemento para su funcionamiento. El fin de documentar las relaciones es facilitar la identificación de los otros elementos afectados en un cambio o fallo.

Por ejemplo, si una base de datos cambia su puerto de escucha, es importante saber dónde se debe cambiar esa información para que sigan funcionando las aplicaciones que consuman la base de datos.

Lo mismo debe tenerse en cuenta para cualquier elemento de configuración.

Un elemento puede depender de muchos elementos. Y un mismo elemento puede tener varios dependientes.

Esta es una de las administraciones más difíciles de mantener y que con el tiempo se debe poder alimentar.

4.6.5 Crear un repositorio de Software

Deben guardarse las diferentes versiones del software. Contar con los instaladores de las versiones correctas teniendo el histórico de cambios.

Algunas organizaciones confían en los repositorios en la red. Lo cual no es una buena idea. El software puede desaparecer de la red en cualquier momento y solo darnos cuenta en un fallo.

Por ello se debe tener nuestra propia versión almacenada para poderla recuperar.

4.6.6 Mantenimiento de la CMDB

El proceso de cambios de los elementos debe tener los pasos necesarios para actualizar la CMDB. Esta debe ser actualizada con cada cambio y corresponder con la realidad.

Además de este paso del proceso, deben hacerse auditorías que revisen la correspondencia entre la CMDB y lo instalado físicamente.

4.6.7 Establecer las líneas base

La línea base es el conjunto de elementos de configuración, sus atributos y sus relaciones en un momento dado del tiempo, que hacen posible el funcionamiento de una aplicación.

La línea base estable es la que funciona, un cambio en cualquier elemento cambia la línea base.

Se debe mantener el histórico de estas líneas base con fechas de implementación y de finalización, y una clasificación: estable, inestable.

Toda línea base debe poder ser reproducible en un nuevo entorno, o usarse para la recuperación ante un desastre.

También es útil para el rastreo de errores.

4.6.8 Métricas de gestión de activos

Algunas medidas que puede obtener este proceso son:

Cambios por periodo: mes, semestre, año.
Fecha de la última línea base.
Costo mensual de los activos.
Cantidad de elementos por línea base.
Cambios de configuración, versión.

4.6.9 Resumen de la gestión de activos

La CMDB es el corazón de la gestión de activos. En ella se registran los elementos que son importantes para la operación. A estos elementos se les llama elementos de configuración.

Tener un registro de todo lo necesario para la operación y sus relaciones es indispensable para reproducir la ejecución del software. Y para rastrear errores.

También es necesario que se actualice en esta base de datos cualquier cambio que se realice.

4.7 PROCESO PARA EL SOPORTE Y MANEJO DE ERRORES

Figura 33. Pasos para implementar el soporte
[Elaboración propia]

En este apartado se tratarán los fallos encontrados en producción; es decir en el entorno operativo del software. Es muy importante cuantificarlos porque da una medición indirecta de la satisfacción del cliente, y de la efectividad del proceso de calidad. Es muy probable que deba incluirse un indicador de soporte en el cuadro de mando integral.

4.7.1 Definir clasificación de los fallos

Hay muchas definiciones para los fallos del software. En el presente se definen, basados en ITIL [3]:

Error: Comportamiento inesperado del software. Una excepción no manejada o una pantalla no esperada serian ejemplos de errores.

Defecto: No hace de forma correcta lo que se espera debe hacer. Un cálculo matemático mal hecho. No imprime un reporte o lo hace de forma defectuosa.

Incidente: Interrupción del servicio prestado por el software. Aquí también caben los incidentes de seguridad.[9]

Lo fallos deben tener un manejo estandarizado y predecible, que permita resolverlos en el menor tiempo posible según su prioridad y de la forma más eficiente. La recomendación de los marcos de procesos es la implementación de una mesa de servicios.

La mesa de servicios recibe la notificación del fallo por medio de un ticket. Revisa si se trata de un error conocido al cual le puede dar manejo, si no puede hacerlo, por no tener el conocimiento técnico o la autoridad, lo escala para un manejo de resolución de problemas.

Este problema es atendido por el personal técnico que tenga el conocimiento para hacerlo y puede dar una solución, caso en el cual se puede convertir en un error conocido. Se debe documentar la solución dada para futuros errores similares.

Si se encuentra el origen del fallo, puede convertirse en una solicitud a desarrollo de un error o defecto, que será priorizada según lo que decida la administración. Si el origen no se encuentra, se queda en el grupo de errores conocidos.

Un proceso genérico de tratamiento de fallos se muestra en la figura Proceso genérico de soporte.

[9] También ITIL define los problemas, que en la presente memoria se les llama errores conocidos.

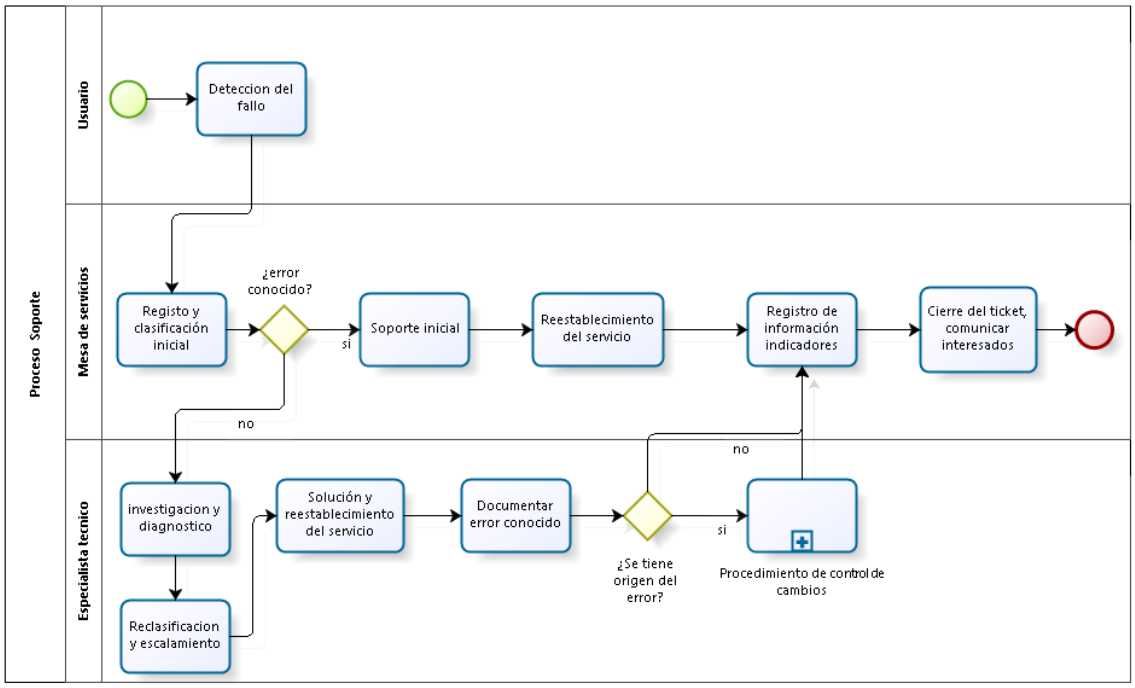

Figura 34. Proceso genérico de soporte.
[Elaboración propia]

4.7.2 La mesa de servicio

La mesa de servicio es un lugar centralizado para atender los fallos del software. En general, usa un software de tickets de soporte para atender las solicitudes de los usuarios. Es posible que también tenga otros medios para recibir las solicitudes: correo electrónico o teléfono.

Debe tener atención en las horas de operación del software. Y su misión es atender los fallos en el servicio que prestan las aplicaciones. Y orientar en el uso del software.

Un ticket puede ser rechazado, solucionado o escalado. Esto último si el técnico que recibe el ticket no cuenta con el conocimiento o las habilidades para resolverlo.

También es importante la clasificación inicial que le dé al ticket, un incidente debe ser atendido de inmediato con la mayor prioridad. Mientras que un error o defecto puede tener un manejo con una prioridad menor.

Es indispensable que la mesa de servicios lleve indicadores. Ya que estos dirán acerca de las prioridades que se deben dar a las soluciones. El análisis de repetición de fallos y los tiempos de respuesta y solución miden el rendimiento del área de soporte.

4.7.3 Definir Especialistas técnicos

Es un rol genérico para quien evalúe el fallo. Es una persona con gran conocimiento del software. Uno de los más serios candidatos es el líder de desarrollo, o alguien del equipo de calidad. Esta persona puede dar un diagnóstico y escalar al personal correspondiente.

Un desbordamiento de memoria es un diagnostico que se puede solucionar con un reinicio. Pero no es el origen del problema. Por lo que escalarlo es importante.

El especialista debe hacer lo posible por reproducir el error en ambientes controlados como pruebas o desarrollo, documentarlo e informar lo encontrado. Si encontró una solución temporal debe documentarla en la base de conocimientos para futuros fallos similares. Esto adicional al escalamiento que haga.

4.7.4 Definir Escalamiento

El fin de escalar el fallo es encontrar una solución temporal y/o definitiva. Casi todos los fallos del software se pueden rastrear hasta diseño y desarrollo de código. Otros, hasta problemas de configuración.

Una persona a la que se le ha escalado un fallo debe hacerse preguntas como estas:

¿Ha habido algún cambio reciente?

¿El error es reproducible en ambientes controlados?

¿Cuántas veces se ha presentado este fallo?

¿Qué tan importante es para el negocio la solución del fallo?

Las respuestas a estas y otras preguntas que surjan de su experiencia y conocimiento darán las pistas que lleven al origen del fallo.

Una vez encontrado el origen lo más probable es que se convierta en una petición de cambio. Puede ser desde un cambio en el hardware, hasta una modificación del diseño o del código, lo que originará un parche, que debe ser implementado en el proceso de cambios.

4.7.5 Documentar las soluciones temporales

El proceso puede encontrar, por ejemplo, que un reinicio restablece el servicio. Pero esto es una solución temporal. También puede encontrar errores en los datos. Bien sea que el software haga un registro deficiente de los datos o que sean cargados de forma errónea por los usuarios. Un script de base de datos puede solucionar este problema.

Puede encontrar que la tarea se puede hacer de forma diferente.

Todas estas soluciones deben ser documentadas y capacitar al personal de la mesa de servicios para su ejecución.

Y todo el trabajo de la mesa de servicio debe ser medido.

4.7.6 Métricas de soporte

Las métricas de la mesa de servicio son de las más importantes para la operación. Son el insumo para la toma de decisiones acerca de las prioridades en las solicitudes de cambios. Y miden bastante bien la satisfacción del usuario.

Se deben llevar estadísticas de los tipos de fallos que se presentan, su cantidad y clasificarlas en orden de aparición

En 4.4.3.2 se dio un manejo de errores del área de desarrollo aplicando la regla de Pareto. Este mismo método se puede aplicar a los fallos de producción. Igual que en

desarrollo, el 20% de los fallos produce el 80% de lo que se ocupa la mesa de servicios, por lo que darles solución definitiva reducirá el soporte. Y nuevos errores reemplazaran a este 20%.

Otra medida que se debe llevar es la disponibilidad.

Se mide con la fórmula:

$$\text{Disponibilidad} = \frac{\text{TMPF}}{\text{TMPF} + \text{TMPR}} \text{ X } 100$$

Donde:

TMPF: Tiempo medio para la falla
TMPR: Tiempo medio para la reparación

Es importante que la unidad de medida del tiempo sea la misma, si es horas que siempre sean horas, también se puede llevar en semanas, días o meses.

Otras métricas de soporte y manejo de errores

Repetición de incidencias.
Cantidad de solicitudes con dudas en el uso del software.
Cantidad de solicitudes de configuración.

4.7.7 Resumen del soporte y manejo de errores

Centralizar la atención de los fallos y las dudas acerca del manejo del software es muy bueno para la operación. Esta centralización se hace en una mesa de servicios que recibe solicitudes a través de un software de administración de tickets. También se pueden recibir las solicitudes por otros medios como teléfono o correo electrónico.

Cada solicitud debe ser evaluada y clasificada. A partir de esta clasificación se debe atender. La solución puede ser un procedimiento manual, como un reinicio o una petición de cambios al área de desarrollo. De todo esto se deben llevar métricas que ayuden a priorizar el trabajo de desarrollo.

Los errores conocidos, la configuración y el manejo del software debe ser almacenado en una base de conocimientos. De lo que se tratará en el próximo capítulo.

4.8 PROCESO DE GESTIÓN DEL CONOCIMIENTO

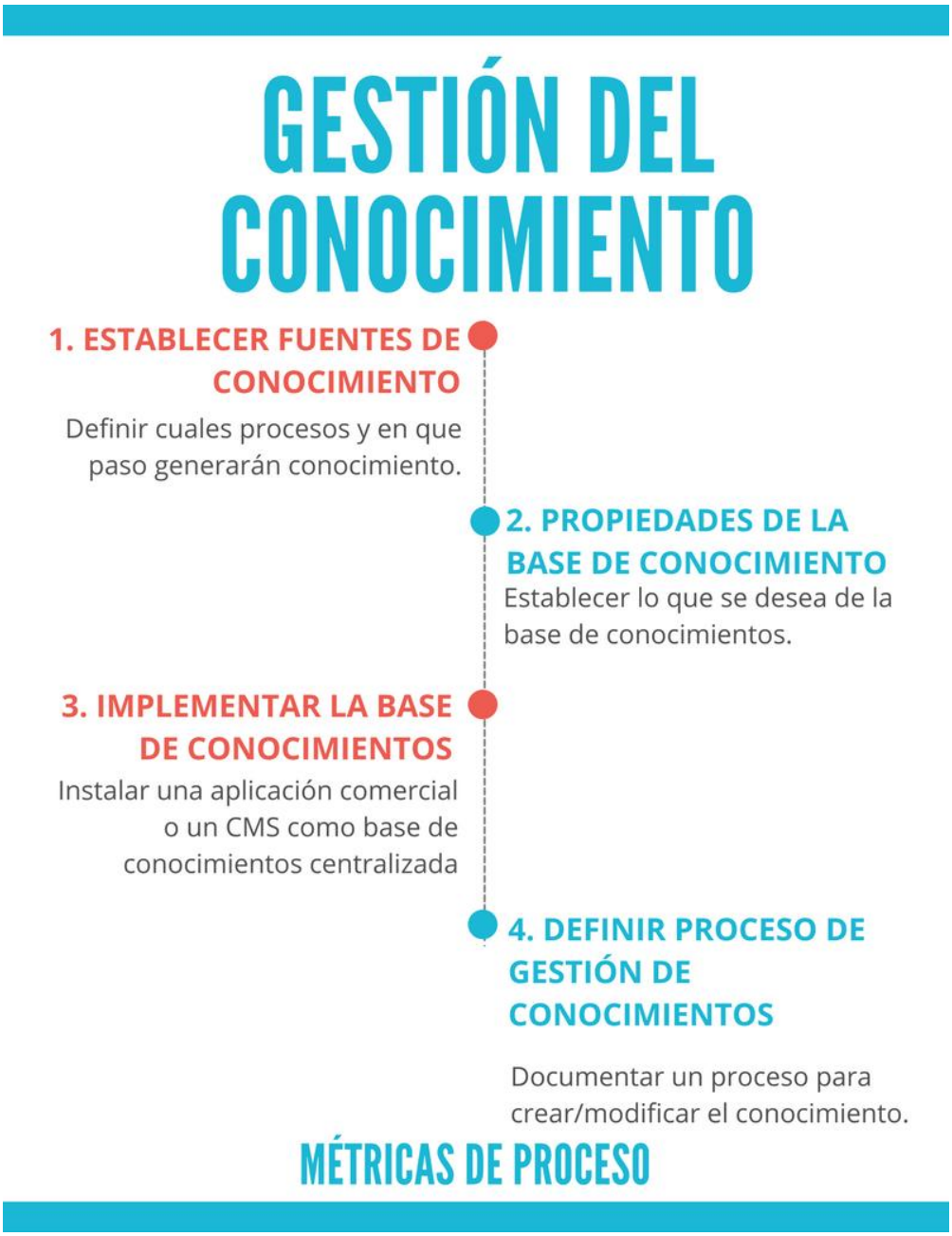

Figura 35. pasos para implementar gestión del conocimiento [Elaboración propia]

La importancia del conocimiento radica en que entre más se usa el software más información se produce. Por ello se debe almacenar y usar.

4.8.1 Establecer fuentes del conocimiento

En la operación de una empresa se genera información, y el procesamiento de la información es conocimiento.

La configuración del software, el manejo de errores conocidos, casos de soporte, forma de actualizar el software. Manejo de herramientas, cómo construir o modificar el programa. Son conocimiento generado en la operación.

Erwin Daniel Suarez Rodriguez. El gobierno del software en fase de mantenimiento

Y este conocimiento debe ser capturado, organizado y almacenado.

Debe existir un proceso documentado para la gestión del conocimiento que incluya la captura, consulta y modificación del conocimiento. El establecimiento de las revisiones pares da una garantía de la calidad de lo almacenado.

Este es un proceso de apoyo dentro de la operación de TI. En ITIL la gestión del conocimiento es un proceso de Transición del Servicio y está a cargo del Gestor del Conocimiento. [3]

La figura muestra la generalidad del proceso:

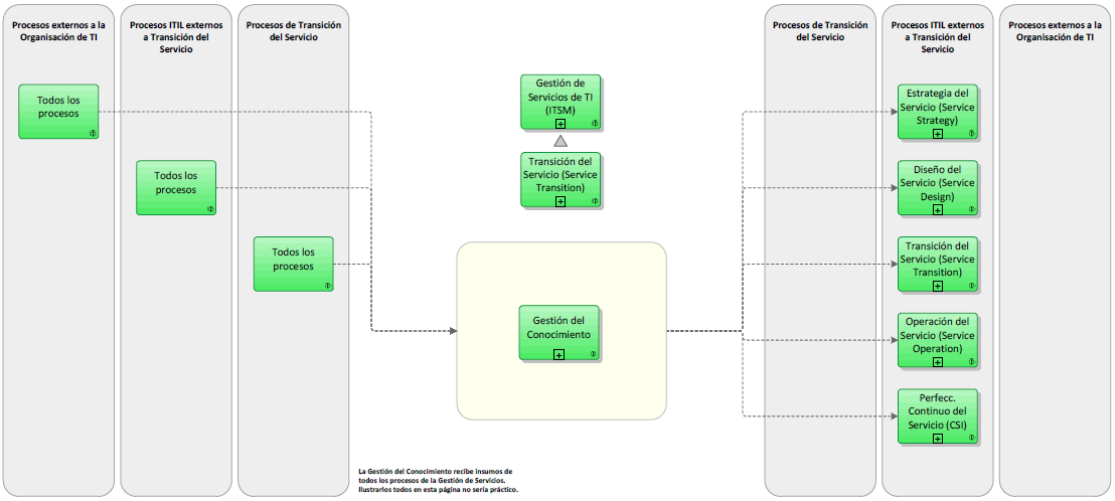

Figura 36. Gestión del conocimiento. [3]

En esta figura se muestra una base de conocimientos centralizada y que se alimenta de todos los procesos. Una de las formas prácticas de manejarlo es hacer llamados desde todos los procesos al de gestión del conocimiento.

4.8.2 Definir propiedades de la base de conocimiento

Las bases de conocimientos son toda una teoría de sistemas. Buscan la recolección, consulta y modificación de conocimientos que son útiles a un grupo u organización humana.

No es del alcance de este trabajo profundizar en el tema. Por lo que se dan las pautas para organizar el conocimiento de una organización con énfasis en el que es necesario para la operación del software, o el conjunto de él.

Una buena base de conocimientos debe contar con al menos:

1. Un lugar centralizado para el almacenamiento de la información.
2. Un sistema de consulta simple.
3. Una taxonomía sencilla para organizar la información.
4. Un soporte informático robusto.
5. Considerar el largo plazo.

Existen en el mercado múltiples herramientas que ofrecen gestionar el conocimiento. La organización tomará la decisión que más se ajuste a su necesidad y presupuesto. Porque más importante que el software usado, es el proceso.

El largo plazo es una de las grandes preocupaciones de los gestores de conocimiento. Nada garantiza que habrá soporte futuro o que un formato de hoy pueda leerse o verse en unos años. Por lo que la base de conocimientos debe ser actualizada frecuentemente a la tecnología de punta. Por ahora no hay una respuesta definitiva a la duración de la documentación informatizada en el futuro.

Como una solución para equipos medianos y pequeños existen dos aplicaciones informáticas que son baratas de implementar: Wikipedia y WordPress.

4.8.2 Implementar la base de conocimientos

Organización de una base simple, wiki o WordPress

Wikipedia se ha convertido en una base de conocimientos de facto para los usuarios de Internet. Tiene millones de artículos que se enlazan unos con otros, además de tener versiones en múltiples idiomas.

Se basa en la cooperación, miles de “wikipedistas” alrededor del mundo crean, revisan y editan contenido a diario.

El software que soporta a esta conocida web es de código abierto y puede ser instalado en un servidor con Apache - PHP - MySQL. Es una de las mejores opciones para implementar una base de conocimientos.

Otra opción muy válida es el uso de un CMS[10] o sistema de gestión de contenidos, Wordpress es el CMS más usado por los creadores de contenido de la red, por lo que tiene un amplio soporte por una gran comunidad de usuarios. Drupal y Joomla son también opciones muy maduras para tener una base de conocimientos.

Estas herramientas ofrecen muy buenas ventajas:

1. Son de código abierto
2. Tienen una amplia comunidad de usuarios en Internet
3. Permiten la búsqueda de información de una forma muy ágil
4. Permiten la colaboración.

Por ello se considera que la elección de un CMS o una WIKI es una excelente opción para un grupo de TI.

4.8.3 Definir proceso gestión de conocimiento

[10] En inglés CMS: Content Managment System

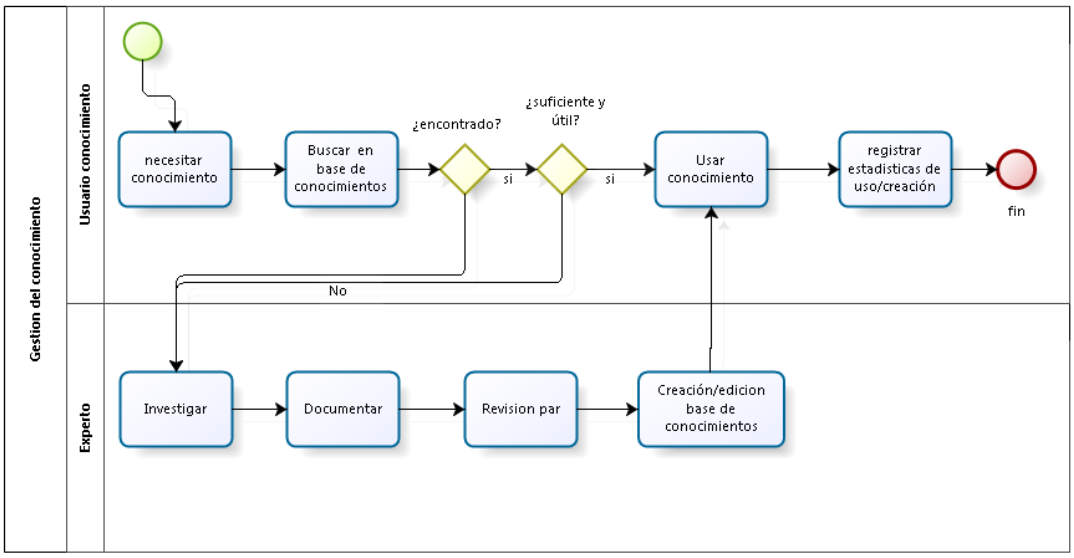

Figura 37. Proceso genérico gestión del conocimiento
[Elaboración propia]

La figura muestra un proceso general para la gestión del conocimiento, este se puede adaptar como un subproceso que se llama desde los demás procesos de modo que se enriquezca continuamente esta base de conocimientos.

La existencia de un proceso y que el personal lo conozca y use es fundamental para el éxito.

4.8.4 Métricas de gestión del conocimiento

Consultas por periodo
Número de artículos modificados
Encuestas de utilidad del tipo ¿Le fue útil la información?

4.8.5 Resumen de gestión del conocimiento

La operación del software produce información útil: casos de soporte, problemas conocidos, soluciones temporales… Toda esta información debe estar al alcance de todo el equipo por lo que debe centralizar y crear un proceso para su almacenamiento, modificación y consulta.

Es un activo que se irá enriqueciendo con el tiempo lo que constituye un activo invaluable para la organización.

4.9 CUADRO DE MANDO INTEGRAL: GOBERNAR EL MANTENIMIENTO DEL SOFTWARE

Figura 38. Pasos para implementar un CMI
[Elaboración propia]

Un conjunto de programas de computadora necesita recursos para operar. Y todos esos recursos deben ser administrados. El enfoque tradicional ha sido administrar el software

como un proyecto, pero esto da muchas dificultades ya que no es posible tener un alcance de lo que se necesita debido a las cambiantes condiciones de la operación. Y adicional, en los proyectos no existe un final fijo en el tiempo.

A los continuos cambios se les suele llamar mantenimiento, y a mantener en ejecución la aplicación, se le llama servicio. En realidad, se trata de la operación del sistema. Mantener el servicio y hacer cambios continuos es un gran problema para la mayoría de equipos de TI.

Pressman [16] lo describe así: *El mantenimiento comienza casi de inmediato. El software se libera a los usuarios finales y, en cuestión de días, los reportes de errores se filtran de vuelta hacia la organización de ingeniería de software. En semanas, una clase de usuarios indica que el software debe cambiarse de modo que pueda ajustarse a las necesidades especiales de su entorno. Y en meses, otro grupo corporativo, que no quería saber nada del software cuando se liberó, ahora reconoce que puede ofrecerle beneficios inesperados. Necesitará algunas mejoras para hacer que funcione en su mundo.*

El reto del mantenimiento del software comienza. Uno se enfrenta con una creciente lista de corrección de errores, peticiones de adaptación y mejoras categóricas que deben planearse, calendarizarse y, a final de cuentas, lograrse. Mucho antes, la fila creció bastante y el trabajo que implica amenaza con abrumar los recursos disponibles. Conforme pasa el tiempo, la organización descubre que emplea más dinero y tiempo en mantener los programas existentes que en someter a ingeniería nuevas aplicaciones. De hecho, no es raro que una organización de software emplee entre 60 y 70 por ciento de todos sus recursos en mantenimiento del software.

Acerca del por qué se necesitan tantos cambios es un tema que ha sido tratado extensamente en muchas publicaciones, el resumen es que todo tiende al cambio, y el software no es ajeno a esto y debe ajustarse. Pero se adiciona que en el proceso de cambio deben seguir ejecutándose los programas.

En el presente estudio se han mostrado diferentes marcos de procesos que se usan con éxito en la construcción o la operación del software. El reto es poner todo esto junto. Ya que el administrador de TI puede tener un conjunto de procesos sueltos que lo arrastren al fracaso. Y siempre, tendrá recursos limitados.

Para tener una operación exitosa se requiere:

1. Ejecutar procesos probados de ingeniería de software
2. Tener una adecuada administración
3. Asegurar la calidad del producto

Los puntos uno y tres han sido desarrollados a lo largo de esta memoria. Indicando los mínimos de los procesos que se deben implementar.

El presente capítulo se encargará del punto dos, proponiendo el desarrollo de un Cuadro de mando integral o CMI. Mostrando los pasos que se deben seguir en su implementación y dando pautas para sacar provecho de esta herramienta.

4.9.1 Generalidades del cuadro de mando integral

El CMI fue introducido a la administración de negocios por Kaplan y Norton [20] desde 1992. Se estableció como herramienta de gestión que alinea la estrategia empresarial

con unos objetivos medibles. Esto, en respuesta a la administración clásica que solo tenía en cuenta la administración financiera como herramienta de gestión.

Además de la financiera, el CMI propuesto por Kaplan y Norton tiene tres perspectivas más, para un total de cuatro, que una buena organización debe medir.

Así, un buen CMI debe medir cuatro perspectivas:

- La perspectiva financiera
- La perspectiva del cliente
- La perspectiva del proceso interno
- La perspectiva de aprendizaje y crecimiento.

Cada una de estas perspectivas tiene unos objetivos. Por lo que las mediciones deben hacerse solo de las metas que se quieren alcanzar o mantener.

Se busca, por ejemplo, que midan el alcance de objetivos financieros, la satisfacción del cliente, calidad de los productos, aprendizaje de los colaboradores, tasa de retención del personal…

Los indicadores de cada organización son diferentes debido a que se buscan diferentes objetivos. El valor del CMI es que el día a día de la organización apunta a la estrategia empresarial.

Las organizaciones han encontrado grandes ventajas en el uso de CMI, por lo que su uso se ha extendido a otras áreas del conocimiento incluyendo TI.

4.9.2 El CMI en TI

Van Grembergen, Van Bruggen [21] fueron los primeros en describir el uso de CMI en TI. Los resultados de los estudios de estos autores generaron un CMI genérico para TI conocido como IT BSC[11].

En este IT BCS, y en similitud al trabajo de Kaplan y Norton. Van Grembergen y De Haes [21] presentan cuatro perspectivas para TI:

1. Perspectiva de orientación al usuario
2. Perspectiva de excelencia operacional
1. Perspectiva de orientación futura
2. Perspectiva de contribución al negocio

La primera perspectiva mide la evaluación del usuario, la segunda mide los procesos empleados para entregar los servicios al cliente, la tercera mide los recursos humanos y tecnológicos necesarios, y la cuarta mide el valor para el negocio de las inversiones en TI.

Patricia Perez y Lourdes García [22], proponen una metodología para construir un CMI de TI. Con una serie de pasos que convierten la estrategia en unos indicadores usables, e incluso, proponen el uso de un indicador único para TI.

A partir de los trabajos enumerados, la experiencia del autor y los procesos que se sugieren en los anteriores capítulos de esta memoria. Se formulan unos pasos para la implementación de un CMI en TI.

[11] Del inglés: IT Balanced Scorecard

Para ello se usará el software BSC Designer[12], los mismos pasos pueden ser realizados para el diseño de un CMI de TI en cualquier organización.

También se puede usar Excel y otras herramientas de software.

4.9.3 Implementación de un CMI en TI

4.9.3.1 Preparar los datos del CMI

Aconsejan Kaplan y Norton [20] que deben crearse una combinación de indicadores de causa efecto; de resultado; inductores de actuación y vinculados a las finanzas

Los indicadores de causa efecto, son los que muestran que los cambios en un valor afectan algo más. Si mejoramos esto sucederá esto otro. Por ejemplo: si se reduce el tiempo de compilación e instalación en pruebas, el equipo de pruebas contará con más tiempo para ejecutar las pruebas funcionales, si tiene más tiempo para probar, el número de defectos se reducirá.

Los indicadores de resultado muestran el rendimiento del proceso: puntos de función por semana, defectos corregidos por mes. Estos se complementan con los indicadores de actuación: defectos por periodo, disponibilidad del sistema.

Y los vinculados a las finanzas muestran valores que se puedan llevar a precios. Requisitos X trabajador, Procedimientos automatizados.

Estos tipos de indicadores deben agruparse en cada una de las cuatro perspectivas. De acuerdo con la estrategia definida por la organización.

Los datos necesarios son: la estrategia organizacional, los procesos, el contrato con el cliente.

4.9.4 El caso de estudio

Como caso de estudio se toma un software usado en una entidad de tránsito del estado en Colombia. Y desarrollado y mantenido por Data Tools[13]. Sirve para la facturación, recaudo y cobro de trámites de tránsito a los ciudadanos, tales como multas de tránsito[14], financiación de las multas, sacar un vehículo de los patios[15] y otros tramites más. Por cada trámite que el ciudadano deba pagar, por el uso del software se cobra una parte de lo facturado de acuerdo con un contrato entre la entidad y la empresa.

La situación inicial es:

1. **Objetivo estratégico de la compañía:**

[12] Disponible en http://www.bscdesigner.com, existe una versión de prueba de treinta días y opción de uso para estudiantes de postgrado con universidades con convenio.
[13] Empresa donde labora el autor, Se dedica al desarrollo de software a medida para la automatización de procesos de negocio. Su principal producto es software para la gestión de tramites de Tránsito. Web: https://www.datatools.com.co/
[14] Las multas de tránsito se llaman en Colombia comparendos, ya que, desde el punto de vista jurídico, es la obligación de comparecer ante la autoridad de tránsito para presentar recursos contra la sanción. En la práctica, la mayoría de los sancionados se acercan a pagar.
15 Cuando un vehículo es retenido por las autoridades es llevado a un lugar administrado por el estado llamado Patios. El ciudadano además de la sanción debe pagar por el transporte y estadía del vehículo en este lugar.

El objetivo que debe cumplir el área de TI respecto a este contrato.

Cumplir al 100% los compromisos contractuales con el cliente.

Data Tools, presta la infraestructura; es decir el centro de datos, su equipo de desarrollo para ajustar el software a la normatividad y a las solicitudes del cliente[16], una mesa de servicio para dar soporte y capacitación en el uso del software en las horas que hay atención en las oficinas y su conocimiento en el negocio. La entidad del estado pone el personal que atiende al público y las oficinas de atención.

2. Los recursos humanos

La empresa destinó a tiempo completo: dos analistas de requisitos; cuatro desarrolladores; dos analistas de pruebas y dos analistas de soporte. A tiempo parcial, que deben atender este y otros contratos: un gestor de servidores; un gestor de bases de datos; un gestor de la configuración; un arquitecto de software y un gestor de red e infraestructura.

Todo este recurso es dirigido por el gerente de Ingeniería de software, quien garantiza el cumplimiento del contrato, administra el personal y tiene a su cargo otros contratos.

3. El software consiste en tres aplicaciones, las cuales se conocen como Circulemos®.

1. Aplicación para los usuarios de la oficina que atienden al público, desarrollado como applets de java, base de datos IBM Informix y se opera en un navegador de Internet. Este software está en operación desde 2001, 16 años al momento de escribir esta memoria.
2. Una página Web donde los ciudadanos consultan el estado de su trámite o verifican si tienen deudas, embargos o sanciones sin pagar. Esta página está desarrollada en PHP versión 5.2. Usa la misma base de datos Informix y corre sobre Apache Versión 2.2. Se encuentra en operación desde el año 2005. 12 años de operación al momento de escribir esta memoria.
3. Un conjunto de servicios web que se comunican con el RUNT[17] e informan que el ciudadano hizo un pago u otro tramite. También hay servicios web consumidos por otras entidades, estos son desarrollados en Java 1.6 y corren sobre Tomcat 5.2.

En la figura, se muestra la arquitectura de alto nivel de Circulemos®, no incluye los servicios web.

[16] Tiene asignadas 200 horas de desarrollo mensuales para solicitar cualquier desarrollo diferente a cambios de ley.
[17] RUNT: Registro único nacional de tránsito. Base de datos centralizada y administrada por el estado donde se almacenan toda la información de tránsito a nivel nacional. Web http://www1.runt.com.co/

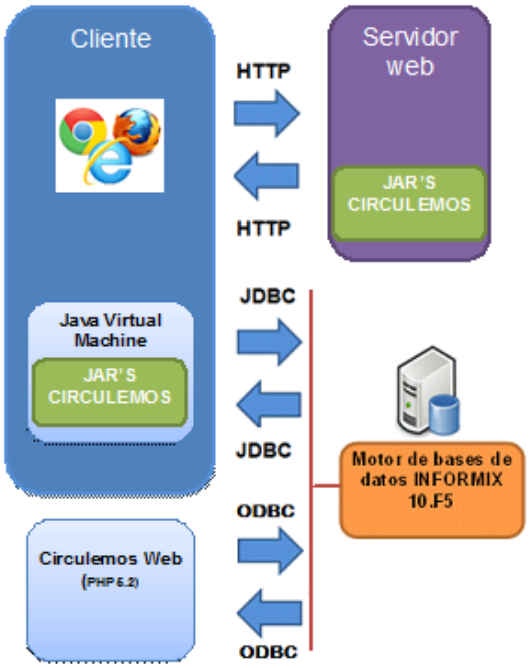

Figura 39. Arquitectura de Alto nivel de Circulemos
[Fuente: Documentación funcional Circulemos]

El Diagrama de los componentes de los servicios web de Runt:

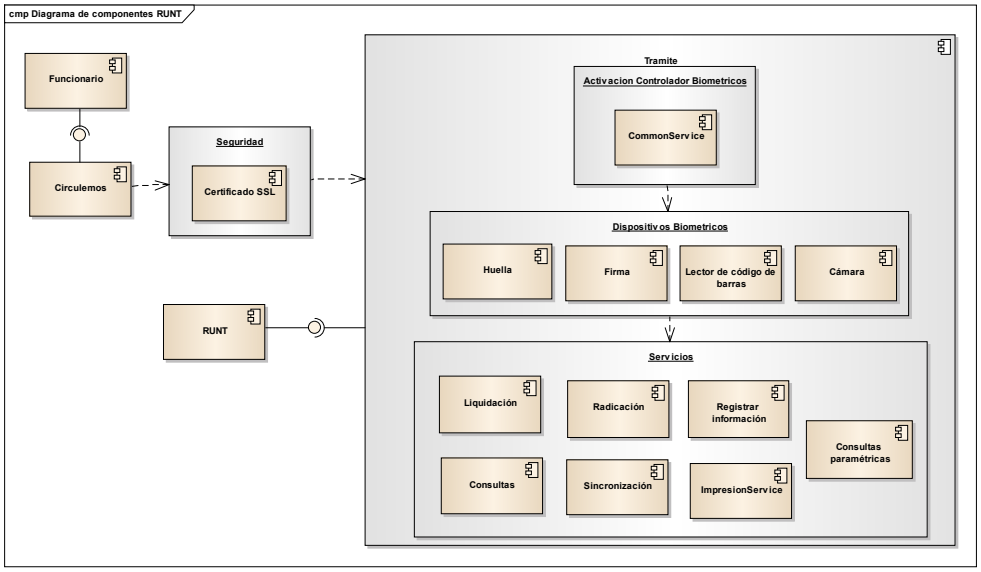

Figura 40. Diagrama de componentes servicios web de RUNT
[Fuente: Documentación funcional Circulemos]

Vista de Circulemos:

Figura 41. Menú inicial de Circulemos para los usuarios que atienden al público vista con todas las opciones [fuente: ambiente de pruebas Circulemos, Data Tools S.A.]

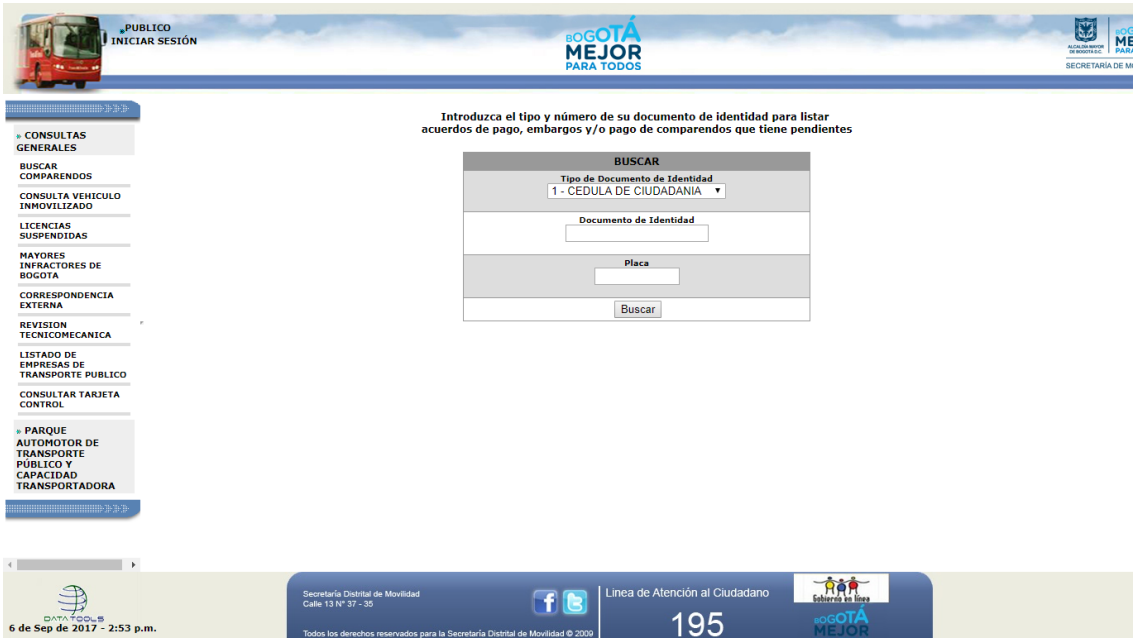

Figura 42. Vista inicial del sitio web de consulta del público [Fuente: http://consultas.transitobogota.gov.co:8083/publico/index.php]

Web Services

Endpoint		Information	
Service Name:	{http://implementacion.wsnotifpagosimit.datatools.com.co/}ServicioNotificarPagoSimit	Address:	http://172.16.5.153:8082/WsNotifPagoSimit/ServicioNotificarPagoSimit
Port Name:	{http://implementacion.wsnotifpagosimit.datatools.com.co/}ServicioNotificarPagoSimitPort	WSDL:	http://172.16.5.153:8082/WsNotifPagoSimit/ServicioNotificarPagoSimit?wsdl
		Implementation class:	co.com.datatools.wsnotifpagosimit.implementacion.ServicioNotificarPagoSimit

Figura 43. Vista del servicio web de notificación de pagos usado por los bancos [Fuente: Ambiente de pruebas Circulemos]

4. Las condiciones del contrato:

Las condiciones del contrato que competen al área de TI son:

a. Disponibilidad del 96.6%. El software debe operar en este porcentaje del tiempo en las horas de atención de la entidad, lunes a sábado de 9:00 am hasta el cierre de la oficina a las 7:00 pm y se atiende a todos los usuarios que estén en la fila. Promedio de cierre de la operación 8:30 pm.
b. Se debe tener disponibilidad de soporte en las horas de atención. El soporte de primer nivel debe ser atendido en máximo dos horas. De segundo nivel en 12 horas y de tercer nivel se negocia la fecha de entrega.
c. El software debe cumplir la normatividad vigente[18]. Y Data Tools es responsable de realizar los desarrollos necesarios para cumplir la ley.
d. El cliente cuenta con 200 horas de desarrollo mensuales para su libre disposición, en las cuales solicita requisitos para el uso del personal de la entidad o mejoras en el servicio. Estos requisitos tienen un cronograma de entrega que debe cumplirse. Y una garantía de 45 días a partir de la puesta en producción.
e. Los errores del software deben ser reparados por Data Tools, según cronograma acordado.
f. Después de una actualización no se puede presentar más de un incidente.
g. Cualquier incumplimiento en estas condiciones será castigado en la facturación mensual.

5. Las condiciones de TI para la ejecución de este contrato.

Cuando se decide darle un manejo estratégico a este contrato las condiciones son:

a. La empresa tiene nivel 3 de CMMI, se maneja el mantenimiento con los procesos evaluados, lo que hace que sea burocrático el mantenimiento del producto.
b. Hay un cronograma de entrega de requisitos en incumplimiento. Este cronograma está a tres años vista. Y constantemente se actualiza y se incumple.
c. El soporte es dado por los analistas de pruebas y si no pueden resolver, por los desarrolladores. Los analistas de soporte se limitan a recibir las solicitudes.
d. Hay una gran cantidad de errores reportados y no resueltos y errores ya resueltos que han vuelto a aparecer. Estos errores no tienen ninguna clasificación.
e. La pila de solicitudes de requisitos nuevos contempla al menos cinco años de desarrollo.

[18] Colombia es un país bastante burocrático, según el canal de youtube VisualPolitik, en 2016 se produjeron 48 normas al día. No todas afectan al software de estudio. Pero es recurrente que haya cambios por cambios en la norma.

f. Los desarrolladores usan cada uno la herramienta que considera conveniente. Hay mezclas de versiones de eclipse, Net Bean, JDK etc. La documentación del producto es poca y está regada por diferentes directorios, correos.
g. Hay un repositorio subversión para el versionamiento de código fuente, pero no tiene ninguna política de uso. Todos suben el código al mismo directorio.
h. Todo el proceso de integración, compilación, y pruebas es manual. Es prácticamente imposible volver a una versión anterior.
i. No hay un inventario de activos del software: servidores, bases de datos, configuraciones. Todo depende del conocimiento del personal.
j. La forma de funcionar el software, de instalarlo, construirlo depende del conocimiento de las personas, las cuales rotan bastante por el estrés laboral que produce la sobrecarga de trabajo.
k. Las pruebas son hechas de forma empírica por los analistas de pruebas.
l. Hay muchos indicadores de proceso, pero ninguno tiene un propósito práctico.
m. El cliente está inconforme y la facturación de la empresa se afecta ya que siempre se descuentan las sanciones por estos incumplimientos.

La situación de este proyecto se resume en la figura:

Figura 44. Situación del caso de estudio
[Elaboración propia]

A continuación, la aplicación de las soluciones, según lo estudiado.

Dadas las condiciones iniciales, se organiza el área según lo que se ha propuesto en esta memoria.

4.9.4.1 El modelo de negocio y la estrategia del caso de estudio

El modelo de negocio es simple, por cada uso efectivo del software que genera un pago, y si el ciudadano efectivamente hace el pago, Data Tools recibe una comisión. El modelo de negocio consiste en maximizar el pago de los ciudadanos.

Para ello Data Tools realizó un convenio con un banco para que tenga un punto de recaudo en la misma oficina de atención, este pago lo notifica el banco al Circulemos a través de un servicio web, una vez pagado el ciudadano se puede retirar de la oficina, ya que Circulemos continuará con su trámite.
Para maximizar este modelo de negocio se debe:

- El software esté en operación todo el tiempo del servicio
- No haya fallos o defectos en producción.
- Cumplir todo el contrato para poder cobrar todo lo facturado.

El modelo de negocio se muestra en la figura:

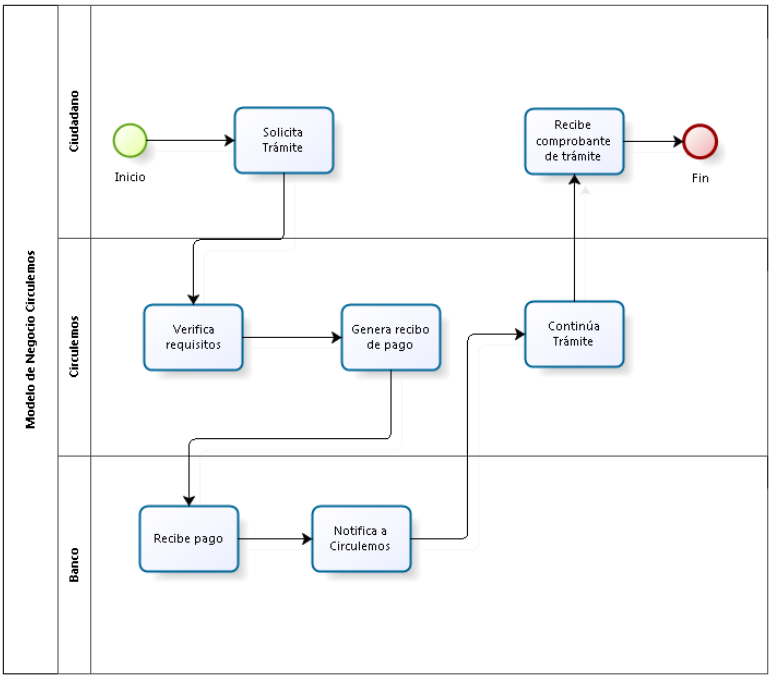

Figura 45. Modelo de Negocio Circulemos
[Elaboración propia con Bizagi Modeler]

El objetivo estratégico de la compañía:

Cumplir al 100% los compromisos contractuales con el cliente.

Llas condiciones contractuales del cliente son el insumo para crear los objetivos estratégicos de TI.

A este conjunto de objetivos se le llama mapa estratégico, para ello se usa el software BSC Designer[19]

[19] Disponible en https://es.webbsc.com/

El grupo de TI toma esta estrategia cómo su insumo, con ella define la estrategia de TI. Para su mejor visualización se crea un mapa estratégico.

Lo que se ha definido es una serie de objetivos de acuerdo con cada una de las perspectivas. Para cumplir cada objetivo se definen indicadores.

4.9.4.2 La gestión de la configuración

Para organizar la gestión de la configuración se procedió a

1. Organizar el repositorio subversión
2. Crear políticas para el uso del repositorio
3. Implementar un servidor de integración continua

El repositorio Subversión se organizó de acuerdo con las recomendaciones del fabricante y aplicando los patrones de diseño detallados en 4.2, así:

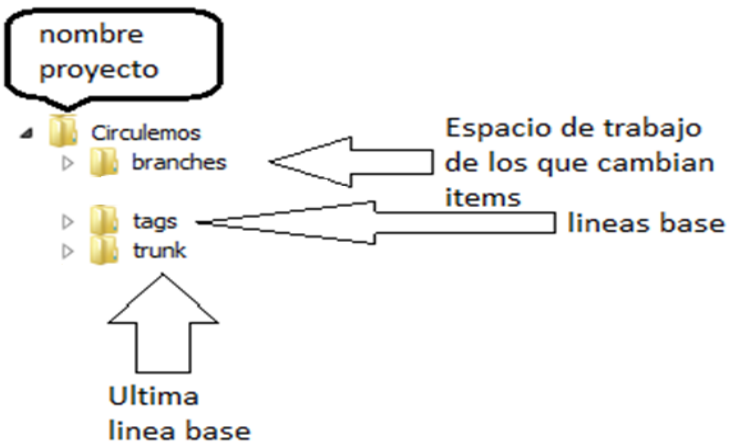

Figura 46. Organización del repositorio
[Elaboración propia]

Las políticas adoptadas son:

El directorio trunk es donde se almacena la última versión revisada del software; es decir la versión de producción. Y según los patrones vistos es la línea principal. O última línea base.

El directorio branches es para que se hagan los cambios. En él se hacen copias del trunk, se modifican y se integran de nuevo al trunk, cuando pasan las pruebas. En los patrones son la línea activa de desarrollo.

El directorio tags, es para conservar copias de las versiones liberadas, en lenguaje CMMI se les llama líneas base.

Se crea un directorio de integración donde se mezclan los diferentes desarrollos de los programadores y un directorio de pruebas donde se almacena la versión que este probando el grupo de calidad.

De forma gráfica esta actividad se ve así:

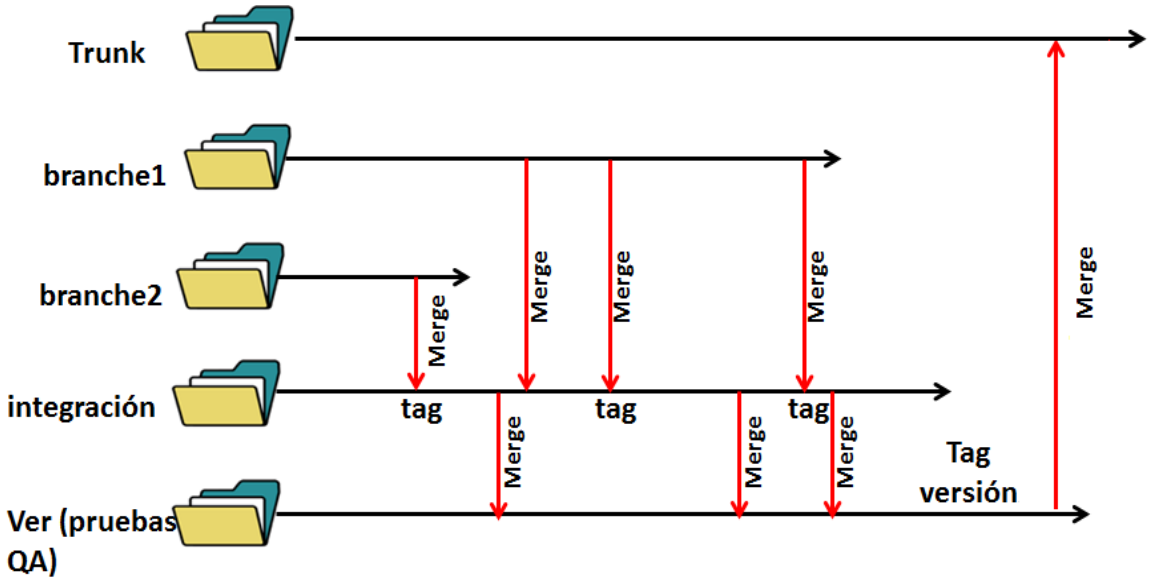

Figura 47. Flujo de trabajo y mantenimiento del repositorio
[Elaboración propia]

Para integración automática se implementó un servidor Jenkins, este hace una revisión estática del código, compila, crea los componentes y los instala en un ambiente de pruebas.

Vista de Jenkins:

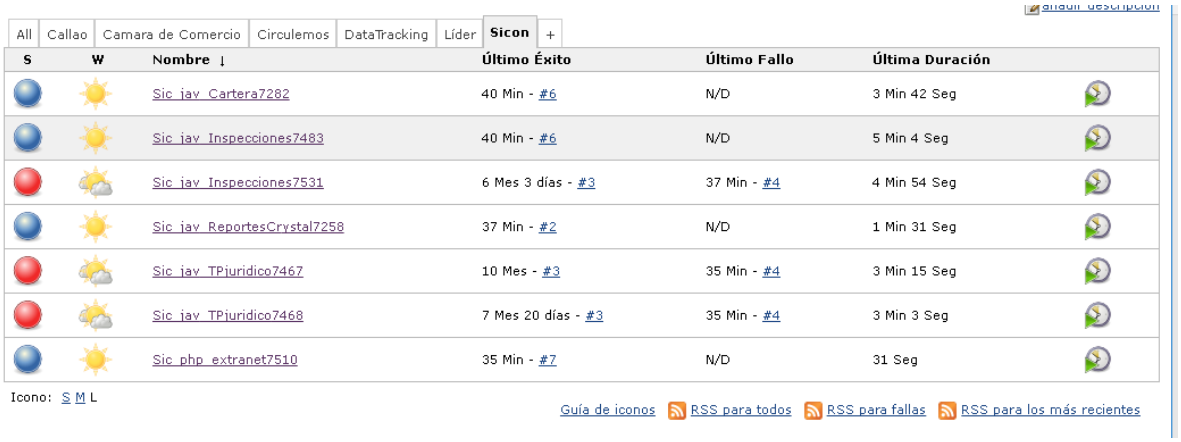

Figura 48. Vista del panel de Jenkins con tareas de integración creadas.
[Fuente: Area de desarrollo Data Tools]

La esfera azul de la derecha quiere decir que la ejecución de la tarea fue un éxito, si es roja es porque fracasó y se debe revisar.

Vista de Sonar, analizador de código estático:

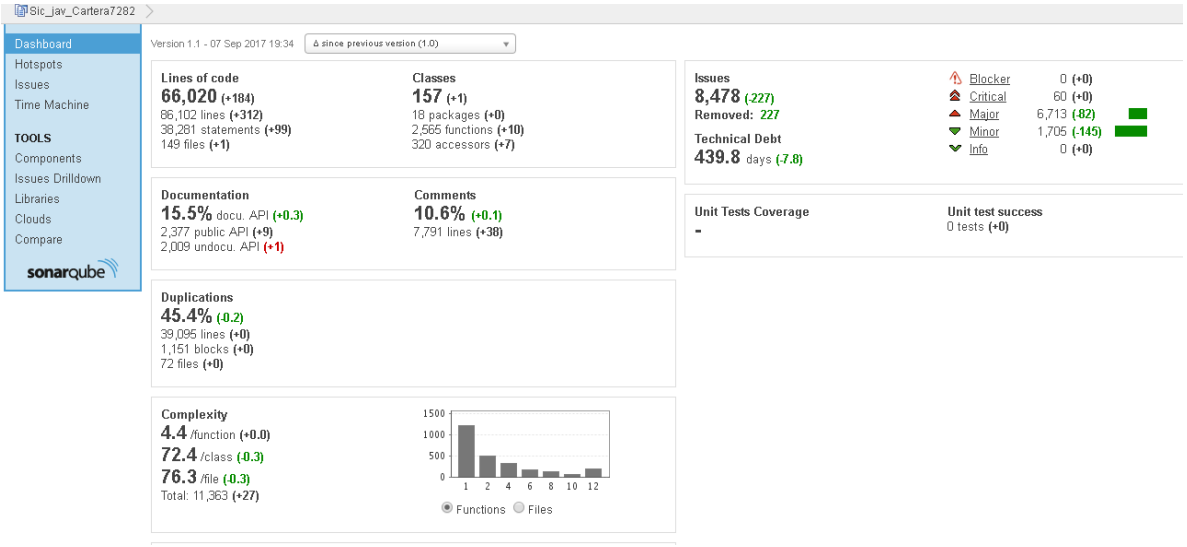

Figura 49. Vista de una inspección de código estático con Sonar
[Fuente: Area de desarrollo Data Tools]

En la parte superior derecha están los tipos de errores que hay, lo que se espera es que los desarolladores no introduzcan nuevos errores, en este ejemplo se evidencia con el numero verde entre paréntesis, que es negativo; es decir que hay menos errores que en la revisión anterior.

Y para finalizar se igualó el ambiente de desarrollo de todos los programadores para que usen el mismo compilador, la misma versión del JDK y los mismos complementos. Esto se dejó en un repositorio y hay instrucciones de cómo crear el ambiente de construcción en la base de conocimientos.

Este proceso es el que ha tenido más impacto en el desarrollo. Las políticas de uso del repositorio han permitido mantener separados los desarrollos, ya que es común que cambien las prioridades, por lo que se integran cuando ya se está seguro de la entrega.

La compilación y análisis automatizado le quitó una actividad a los desarrolladores que antes lo hacían manual y adicional garantiza que los componentes son compilados desde el código fuente que está en el repositorio.

4.9.4.3 Implementación de desarrollo ágil

Como marco de desarrollo se eligió Scrum. Y las políticas dispuestas son:

1. Sprint de dos semanas al final de las cuales se entrega una versión.
2. Las prioridades de cada sprint se negocian con un mes vista. Dos sprint.
3. Lo que se compromete en el sprint en curso no se puede cambiar.
4. Si hay un defecto bloqueante en producción se repara de inmediato.
5. El cliente puede hacer solicitudes en cualquier momento, pero va a la pila de producto.

De esta forma se gestiona la pila de solicitudes a un mes de vista, al hacer entregas más frecuentes hay más tranquilidad de parte del cliente, y el equipo de desarrollo está concentrado en trabajar lo asignado en el sprint.

Para gestionar el tablero de solicitudes se implementó Redmine[20], una herramienta de código abierto para la gestión de desarrollo de software.

Vista de Redmine:

Figura 50. Panel de Sprint en Redmine.
[Área de desarrollo Data Tools]

En la parte izquierda está la pila de solicitudes, en el centro esta lo que se está ejecutando y en revisión y en la parte derecha las tareas ya realizadas. Es muy útil este tablero para que el líder de TI se informe de lo que se está haciendo.

[20] Disponible en http://www.redmine.org/

4.9.4.4 Aseguramiento de la calidad

Para organizar el aseguramiento de la calidad se creó un grupo de calidad el cual tiene las actividades:

1. Establecer las pruebas de aceptación de los desarrollos.
2. Probar las versiones del software. Y reportar los defectos.
3. Probar los defectos de producción reportados por el cliente.
4. Llevar indicadores de fallos y establecer recomendaciones para su disminución.

Para la gestión de casos y planes de pruebas se implementó Testlink, que administra casos de prueba, su reutilización y versionamiento.

Vista de TestLink:

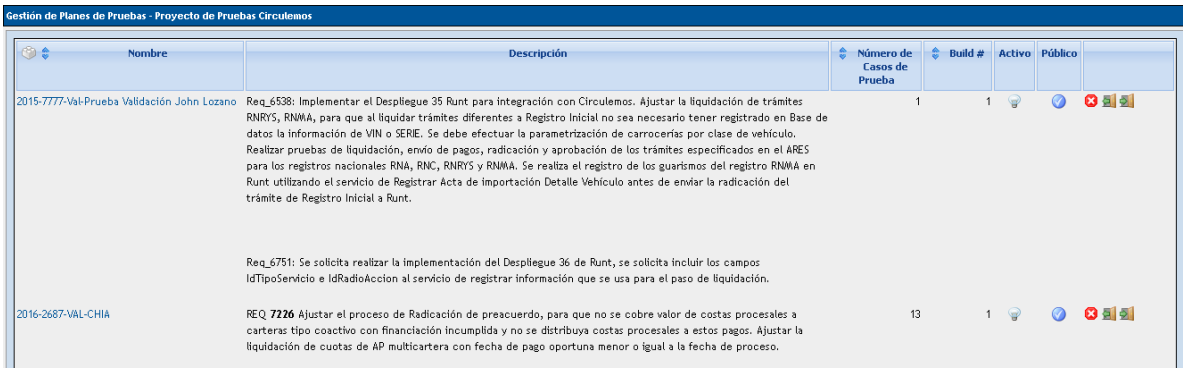

Figura 51. Vista de planes de pruebas en TestLink
[Fuente: Área de Calidad Data Tools]

Para Circulemos debido a la vetustez de la tecnología, no se han podido automatizar pruebas. Se han creado algunas pruebas unitarias, pero es difícil con software que ya está hecho diseñar estas pruebas. Para las pruebas funcionales se ha probado Selenium, sin embargo, no ha sido exitosa la implementación de esta herramienta.

4.9.4.5 La gestión de cambios

Esta actividad no tiene mayor explicación, las reglas establecidas fueron.

1. Se implementará durante el siguiente sprint la versión entregada en el sprint anterior. En horas que no hay operación.
2. Los defectos reportados serán clasificados en defectos, errores e incidentes. Si es incidente se evaluará si se pueden reparar de inmediato, si no es así, se reversará la instalación y la corrección se instalará junto al siguiente sprint.
3. Si el incidente es parte de un despliegue, y no se puede reparar. Se reversará la instalación.
4. Los parches que corrigen defectos serán instalados en cualquier momento, de preferencia fuera de operación y el proceso de desarrollo garantizará que se integre en las próximas versiones.

Este último punto requiere una explicación. Se puede presentar un error en un momento que el desarrollo está muy avanzado. Al presentarse esta situación se saca una copia de la línea principal del desarrollo y se corrige allí. Luego esa corrección debe ser integrada con el resto de la versión que se esté trabajando.

Esta es una actividad tediosa por lo que solo debe ser aplicada si el parche no se puede programar en un sprint, por tratarse, por ejemplo, de un cambio de normatividad o un defecto bloqueante.

4.9.4.6 La gestión de activos

Para la gestión de la mesa de servicios y la gestión de activos Data Tools implementó Aranda Software. El autor no participó directamente en este proceso, por lo que no se proporcionan imágenes.

Las reglas establecidas fueron.

1. Los activos como servidores, licencias, certificados, software están registrados en Aranda Software.
2. Los cambios realizados deben ser autorizados y registrados en la herramienta.
3. Se tiene un histórico de los cambios y de los activos.

4.9.4.7 Soporte y manejo de errores

El soporte se centralizó en la mesa de servicios, lo usuarios vía web ingresan los casos de soporte y se les crea un ticket de servicio, Aranda maneja los niveles de servicio y envía la respuesta por correo electrónico al solicitante.

El proceso establecido se muestra en la figura.

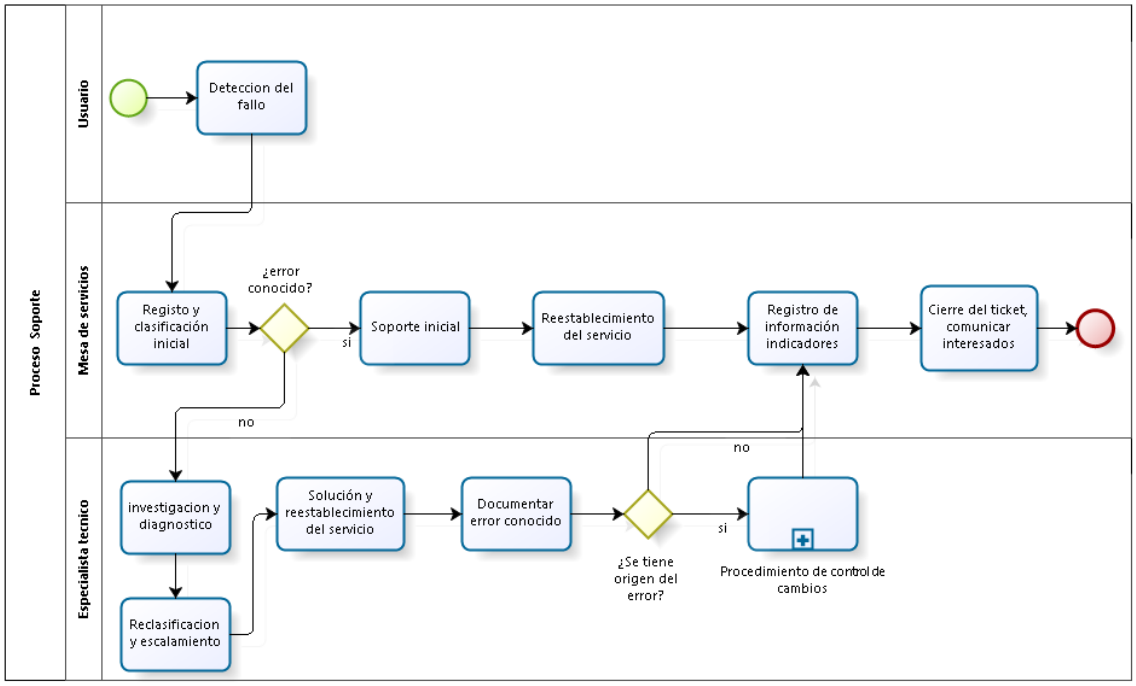

Figura 52. Proceso de soporte.
[Elaboración propia]

Los casos se registran y se les hace seguimiento a través de Aranda:

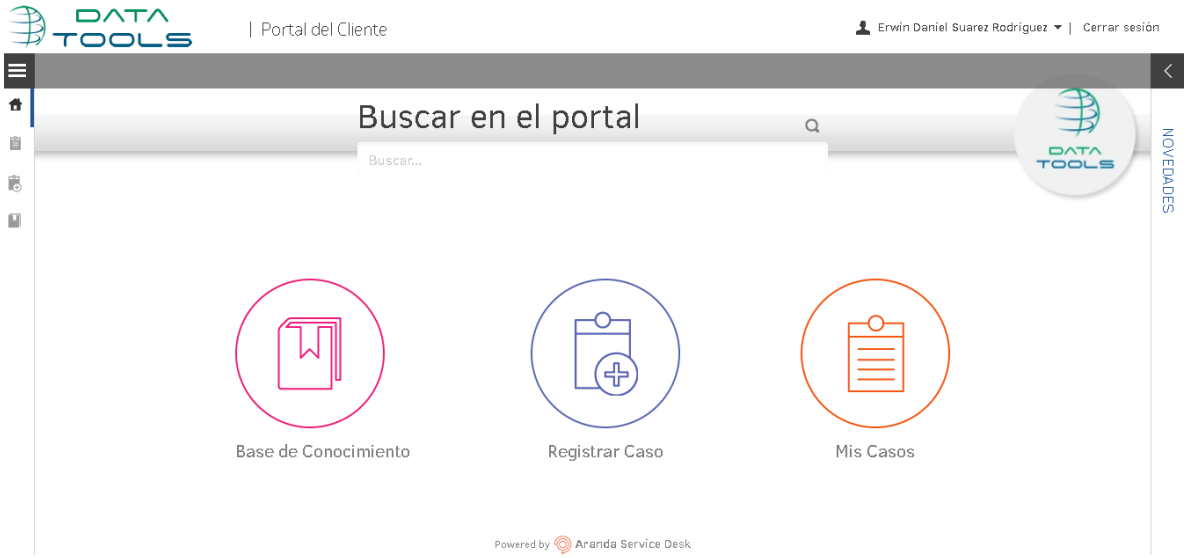

Figura 53. Vista inicial del cliente de Aranda Software.
[Fuente: Web de soporte Data Tools]

En la figura se ve que hay una opción *Mis Casos*, para saber el estado de los casos que haya reportado. *Registrar Caso*, para los nuevos casos y la *Base de Conocimiento*. Donde se almacena información acerca de los errores conocidos del software.

4.9.4.8 La Gestión del conocimiento

Para gestionar el conocimiento se crearon dos fuentes de información. Por un lado, Aranda para todo lo que tenga que ver con soporte del software. Su uso, configuraciones, errores conocidos. Y por otro lado se implementó la wiki de Redmine para lo que tiene que ver con la construcción, modificación y manejo del software.

En la imagen se muestra una parte de un artículo que dice como configurar un ambiente seguro. Por tratarse de applets de java podrían ser descargados y descompilados por un usuario, lo que es un riesgo de seguridad. Para evitarlo se asegura el directorio, y un artículo explica cómo hacerlo. Este es el tipo de conocimiento que está en el conocimiento de algunos y se requiere que cualquiera que lo necesite lo pueda tener.

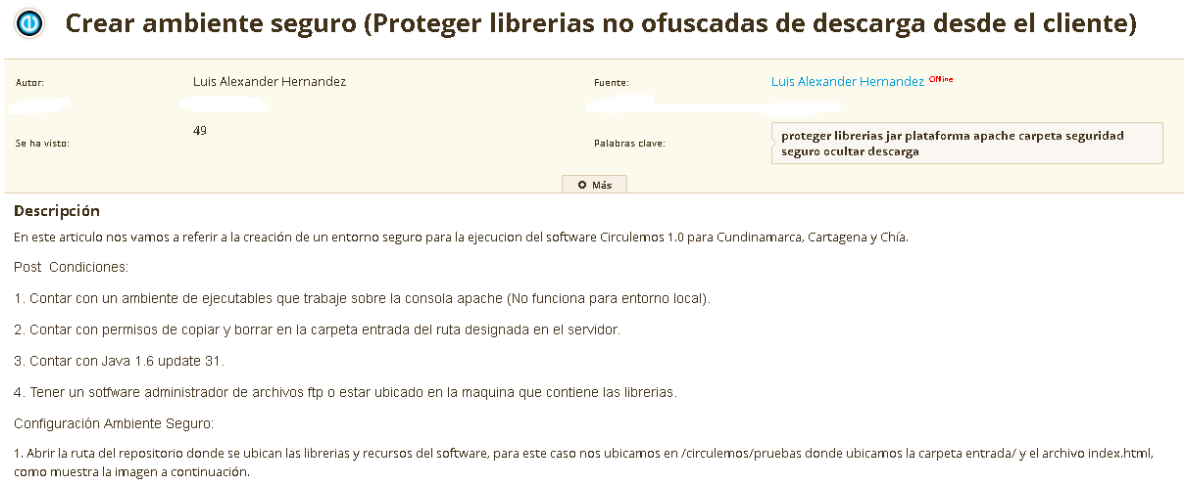

Figura 54. Vista preliminar de un artículo de la Base de Conocimientos de desarrollo de software [Fuente: Wiki de desarrollo Data Tools]

4.9.4.9 Gobernar el mantenimiento

Una vez estén organizadas las áreas de proceso y se tengan los objetivos estratégicos, se procede a crear los indicadores del cuadro de mando.

Para que sean útiles a la organización, los objetivos deben:

1. Contribuir al mejoramiento de los procesos de negocio.
2. Contribuir a alcanzar objetivos organizacionales de la estrategia.
3. Ser alcanzables y técnicamente viables.

Lograr una estrategia de TI es un trabajo de todo el equipo encabezado por la dirección. Es algo que llevará reuniones, opiniones, lluvias de ideas… Lo importante es que lleve a objetivos realistas y que sean medible su progreso. De lo contrario no será posible convertir los objetivos en indicadores.

En el caso de estudio se crearon once objetivos estratégicos. Que se muestran en la figura.

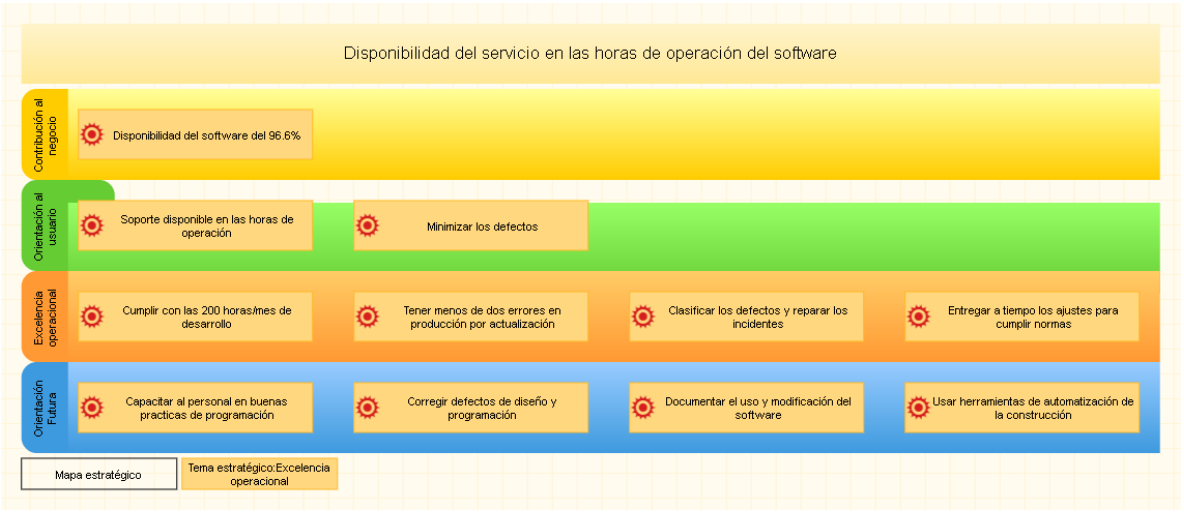

Figura 55. Mapa estratégico TI Data Tools
[Elaboración propia]

4.9.4.9.1 Convertir los objetivos estratégicos en indicadores

Cada uno de los objetivos debe ser convertido en un número según el tipo de indicador que se desee: de causa efecto, de resultado, Inductor de actuación o financiero. Cada objetivo puede tener uno o más indicadores.

También se debe definir la fuente de información, el periodo a medir y las unidades. Lo más practico es realizar esto en una tabla.

Siguiendo con el ejemplo se generan los siguientes indicadores:

Tabla 2. Indicadores de ejemplo Data Tools

	Objetivo	Medicion	Indicador	Tipo de indicador	Fuente de datos	Periodicidad de la medicion	Meta	Acciones
1	Disponibilidad del software de 96,6%	Tiempo que esta disponible para su uso el software en las oficinas de atención al ciudadano	TMPF Disponibilidad = ——— X 100 TMPF + TMPR donde: TMPF: Tiempo medio para la falla TMPR: Tiempo medio para la reparación	de resultado	Mesa de servicio	mensual	97%	un resultado menor a 97% debe aumentarse el tiempo para la falla y reducir el tiempo para la reparación
2	Minimizar los defectos en produccion	Cantidad de defectos producidos tras el despliegue	Promedio de defectos por despliegue	inductor de actuacion	Mesa de servicio	mensual	1	Si hay mas de un defecto se debe rastrear hasta su origen los defectos y mejorar las pruebas, el diseño la programación
3	Minimizar los defectos en produccion	Cantidad de defectos producidos tras el despliegue	Promedio de defectos por despliegue	inductor de actuacion	Mesa de servicio	mensual	1	Si hay mas de un defecto se debe rastrear hasta su origen los defectos y mejorar las pruebas, el diseño la programación
4	Cumplir las 200 horas/mes de programacion	Cumplimiento del sprint	Puntos de funcion desarrollados ----------X 100 puntos de funcion programados	inductor de actuacion	Reuniones de retrospectiva de sprint	mensual	100%	Si no se cumple, se debe verificar si se esta programando mal el sprint y mejorar o si el personal no esta dedicado al sprint
5	Documentar el uso y modificacion del software	Cantidad de articulos documentados nuevos y actualizados en la base de conocimientos	Reglas nuevas * 0,7 + reglas actualizadas por 0,3	causa efecto	Base de conocimientos	mensual	15	faltan mucha documentación (no se sabe cuantas), la meta es documentar 15 al mes. Las nuevas tienen un valor del 70% en el total y las actualizaciones 30% Si no se cumple se deben incluir metas personales en la evaluacion del desempeño

[Elaboración propia]

Se crean los indicadores y se asocian a cada objetivo, en la herramienta BSC Designer se ven así:

Nombre	Iniciativas	Desempeño	Progreso	Valor	Referencia	Objetivo
Balanced Scorecard		37.5%	108.87%			
Contribución al negocio		96%	98.97%	96	0	97
Disponibilidad del software Actualizar mensualmente en el día 1		96%	98.97%	96	0	97
Orientación al usuario		2%	200%	2	0	1
Promedio de defectos por despliegue Actualizar mensualmente en el día 1		2%	200%	2	0	1
Excelencia operacional		97%	97%	97	0	100
Cumplimiento del Sprint Actualizar mensualmente en el día 1		97%	97%	97	0	100
Orientación Futura				0		
Cantidad de artículos documentados por mes Actualizar mensualmente en el día 1		7%	46.67%	7	0	15

Figura 56. Muestra de los indicadores en la herramienta BSC Designer.
[Elaboración propia]

De forma automática, el plan estratégico se grafica con cada uno de los objetivos definidos:

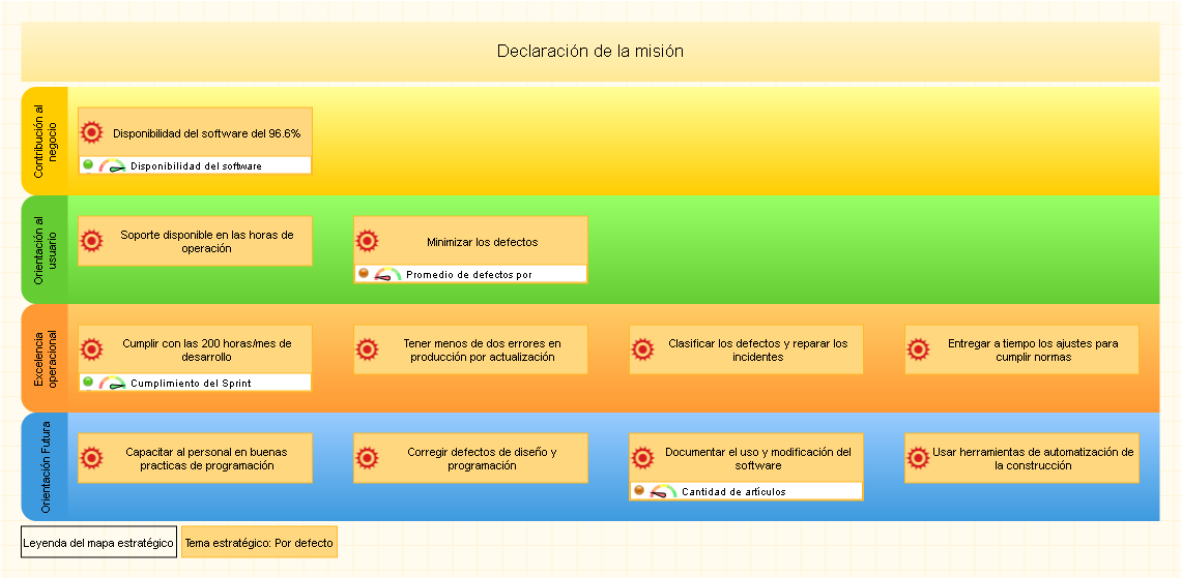

Figura 57. Mapa estratégico con indicadores
[Elaboración propia]

Para el ejemplo no se han creado indicadores para todos los objetivos. En un ejemplo real se deben crear. Se puede empezar con unos hasta tener todos los indicadores. La recomendación es que no sean más de veinte.

4.9.4.10 Crear CMI visual

A partir de estos datos debe crearse visualmente el cuadro de mando, El CMI es similar a los instrumentos de un avión. Las diferentes agujas, números, luces y sonidos van informando a la tripulación que debe tomar los correctivos para llegar a su meta. O, que todo está bien y no se debe hacer algo diferente.

El CMI debe estar en una sola hoja. No debe dar dificultad revisarlo, y el administrador debe hacer lo posible por hacerlo parte suya, y detectar que algo está mal con solo leer el número.

Es muy importante que la fuente de información esté disponible y que se alimente con frecuencia la información. Un buen cuadro de mando debe cumplir las siguientes características:

1. Presentarse en una sola hoja. Una presentación sencilla hace que sea fácil de analizar
2. No tener más de veinte indicadores. Los indicadores deben ser simples y de ser posible tener indicadores visuales. Los gráficos de velocímetro o colores de semáforo ayudan a ver cuánto se está cerca de la meta.
3. Estar alineado con la estrategia del negocio. Esto se ha explicado en los puntos anteriores, debe recalcarse porque puede perderse la utilidad del CMI si no hay objetivos de TI que lleven al cumplimiento de la estrategia organizacional.
4. Un alto compromiso de la alta dirección. Se requiere para que aporte los recursos, impulse las políticas y apoyen las evaluaciones del CMI.

Continuando con el ejemplo, la herramienta BSC Designer plantea un panel de control que de forma muy visual muestra los indicadores.

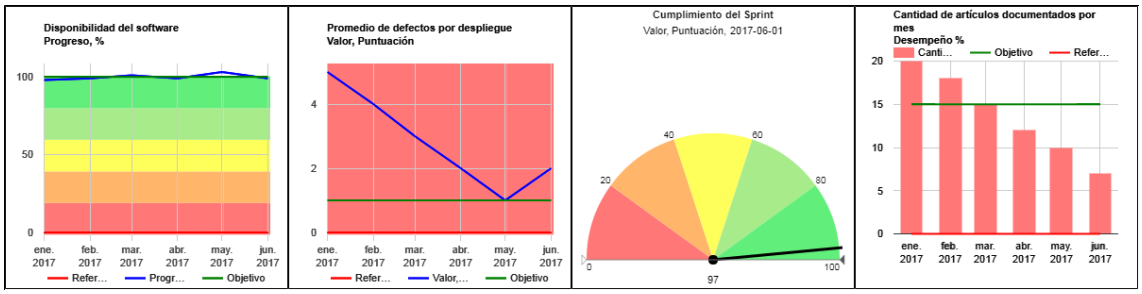

Figura 58. Cuadro de mando integral
[Elaboración propia]

En la figura se muestran cuatro indicadores. Y de forma inmediata se puede ver lo que está bien y lo que no:

- El primer indicador Disponibilidad del software está por debajo del rango meta. Solo en mayo estuvo por encima y en junio volvió a bajar.
- En el segundo indicador, los defectos por despliegue han venido bajando, por lo que la acción tomada ha sido efectiva. Sin embargo, en junio volvió a incumplirse la meta.
- El cumplimiento del sprint no ha llegado a la meta, está cerca pero aún no lo ha hecho.
- La documentación de la base de conocimientos ha venido bajando con el pasar del tiempo.

Con esta información debe trabajar el jefe de TI y tomar acciones. A eso es lo que se llama gobierno.

4.9.4.11 Cómo usar el Cuadro de mando para gobernar el mantenimiento.

El principal objetivo de gobernar un software en operación con un cuadro de mando, es lograr que los objetivos del área de TI, aporten al alcance de la estrategia organizacional. Y esta a su vez, esté alineada con la visión de la organización. De esta forma los recursos limitados con que cuenta la organización se reflejarán en beneficios para el negocio y los clientes. O en el caso de organizaciones que no son negocios, beneficios para el producto/servicio y los usuarios.

Para gestionar con efectividad se sugieren cuatro actividades [20] :

1. Comunicar la estrategia e instrucciones.
2. Asignar recursos.
3. Definir objetivos e instrucciones individuales de equipo y de área.
4. Proporcionar retroalimentación.

4.9.4.11.1 Comunicar la estrategia e instrucciones

Se debe comunicar a cada miembro del equipo, que es lo que hace la organización y cuál es la forma de que hace llegar el producto/servicio a los clientes/usuarios. Debe

formar parte del proceso de inducción y reforzarse en capacitaciones programadas. Lo que para muchas personas es obvio, no lo es para todos. De esta forma se aclara como es que el software contribuye a alcanzar los objetivos de la organización.

El jefe de TI debe dar instrucciones precisas a su personal, según el rol que tenga dentro del equipo de lo que debe hacer para alcanzar los objetivos.

La más grande dificultad que se puede encontrar en este paso es no poder traducir la visión y la estrategia a términos que puedan ser comprendidos. Por ello antes de implementar un CMI, es necesario que exista consenso entre los directivos de lo que significa realmente la visión y la estrategia.

El mapa estratégico es una herramienta muy útil para comunicar este punto.

4.9.4.11.2 Asignar recursos

Un sistema de gobierno estratégico como el propuesto requiere que se asignen recursos humanos, financieros y tecnológicos. Si la organización tiene un presupuesto desvinculado de la estrategia, es poco probable que se tenga éxito en este sistema de gestión.

Cada objetivo debe tener asignados recursos. Y la gestión del cuadro de mando sugerirá lo que se requiere. Por ejemplo, si se tiene la capacidad de construir 200 puntos de función por mes y para alcanzar la meta se requieren 500. Hay que asignar mayores recursos a desarrollo para que alcance el objetivo.

Es deber del gerente de TI asignar estos recursos.

4.9.4.11.3 Definir objetivos e instrucciones individuales de equipo y de departamento

Las evaluaciones de desempeño se han convertido en una herramienta ideal para la negociación de objetivos individuales. Sin embargo, estos suelen ser a corto plazo y muy tácticos. Aunque se debe usar la negociación de objetivos individuales es mejor negociar objetivos estratégicos a mediano y largo plazo.

La llegada de la generación millenials[21] al mercado de trabajadores y su inestabilidad laboral hacen que sea poco aconsejable y difícil de seguir una estrategia a largo plazo. Sin embargo, el administrador debe saber que cada miembro del equipo cuenta, y la asignación de objetivos individuales es indispensable.

También, como equipo y departamento deben asignarse metas. Kaplan y Norton desaconsejan vincular los planes de compensación al cumplimiento de objetivos. Sin embargo, sugieren ejemplos de empresas que han aplicado esta práctica con éxito.

Ejemplos delgobierno en el caso de estudio

[21] Generación de personas nacidas entre 1990 y 1999.

Se presentó un problema con la disponibilidad del software. La mesa de servicio reportaba que el servicio web de notificación de pagos quedaba fuera de servicio y el cliente lo contabilizaba como falta de disponibilidad.

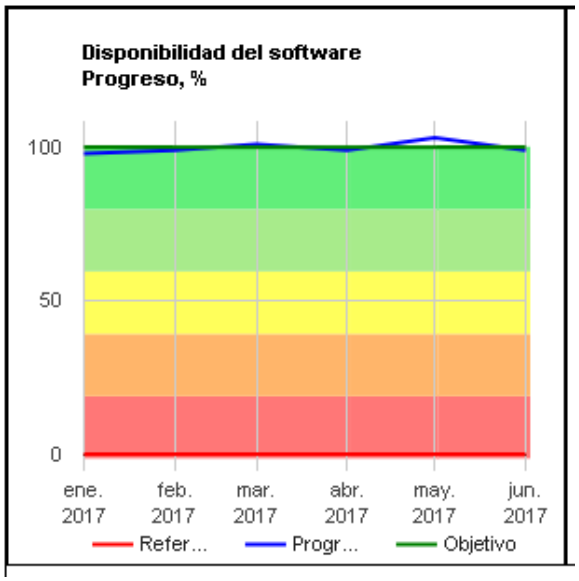

Figura 59. Disponibilidad del software
[Elaboración propia]

El área de soporte sabía que con un reinicio del servicio se resolvía la situación. Por ello se programó un reinicio nocturno. Sin embargo, esto no solucionaba la causa.

El gerente de TI dispuso un desarrollador para encontrar la causa de la falla. Y este encontró que había una conexión a base de datos que se quedaba abierta y agotaba la memoria virtual de java[22]. Se corrigió este defecto del código y dejo de ser necesario el reinicio. Esto no hubiera sido posible sin la gestión hecha por el gerente que asignó recursos y metas específicas. Al parecer este problema tenía años.

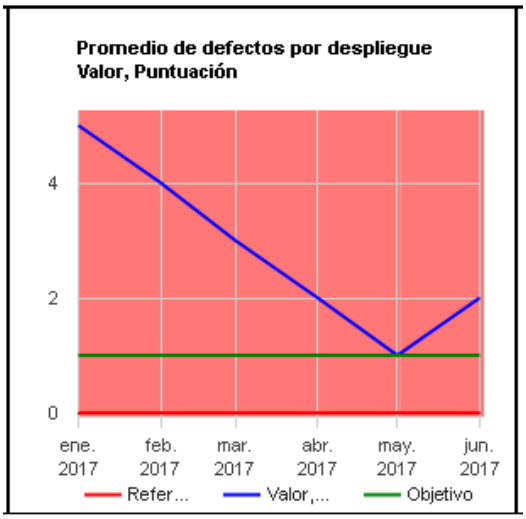

Figura 60. Evolución de los defectos por despliegue.
[Elaboración propia]

[22] El error conocido como java.lang.OutOfMemoryError PermGen space. Se produce cuando una clase no puede ser eliminada por el recolector de basura.

Para la reducción de defectos el grupo de calidad ha documentado más casos de prueba, los defectos se han presentado por situaciones que no se probaron. Por lo que tener una biblioteca de casos de prueba más grande ha resultado.

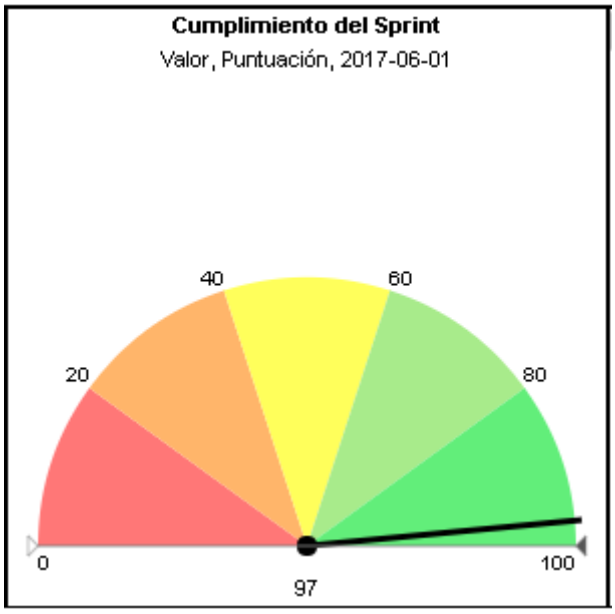

Figura 61. Cumplimiento del sprint.
[Elaboración propia]

El incumplimiento del sprint se ha dado principalmente por la falta de experiencia del personal en trabajar con esta forma de organizarse. El cálculo de los puntos de función ha mejorado, pero aún falta mejorarlo más.

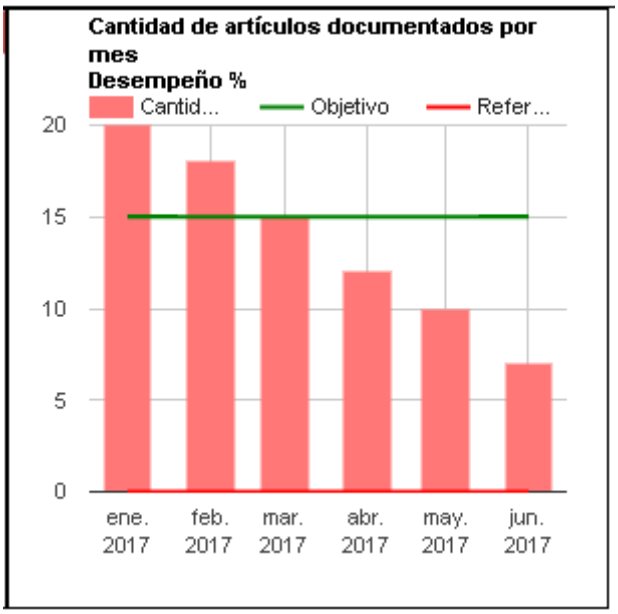

Figura 62. Documentación en la base de conocimientos
[Elaboración propia]

El personal ha argumentado falta de tiempo para documentar la base de conocimientos. En la evaluación del desempeño semestral se puso un objetivo que apunta a esta meta.

4.9.4.11.4 Proporcionar retroalimentación

Se deben hacer reuniones frecuentes para evaluar el cuadro de mando y tomar los correctivos necesarios. Así como algunas fábricas de software creen que, por obtener un nivel de madurez la vida ya está resuelta y no realizan auditorías internas. También es fácil caer en la tentación de creer que la implementación de un CMI es el fin de la historia.

Nombre	Valor	Medida	Ganancia /pérdida	Desempeño	Comparado con ...
Cantidad de artículos documentados por mes	20		13%	20%	7% (7, 2017-06-01)
Promedio de defectos por despliegue	5		3%	5%	2% (2, 2017-06-01)
Disponibilidad del software	95	%	-1%	95%	96% (96%, 2017-06-01)
Cumplimiento del Sprint	70		-27%	70%	97% (97, 2017-06-01)

Figura 63. Vista de semáforo de los indicadores del CMI [Elaboración propia]

En la figura se muestra que los indicadores de cantidad de artículos documentados y el promedio de defectos por despliegue son los que mas se deben trabajar. Son los candidatos que mostrar en una retroalimentación al grupo.

El CMI propuesto es un organismo vivo, que depende de una actividad humana muy intensa: la modificación/corrección/operación de un software. El equipo debe estar en constante alerta para hacer que los recursos limitados den los mejores frutos. Y el CMI es una excelente herramienta para ello.

4.9.4.12 Gestionar los problemas de implementar un CMI

El CMI es una herramienta de gestión, sirve para tomar los correctivos necesarios si hay desviaciones y para orientar el trabajo del área hacia la estrategia organizacional.

Implementar un CMI no es una tarea fácil. Estructurar los procesos definir los indicadores y hacerles seguimiento es una labor ardua y que requiere la implicación directa de las directivas de la organización.

Los gerentes de TI, lideres técnicos, gestores de mesa de servicio y del centro de datos deben estar vigilantes a la gestión del área a partir de los datos del cuadro de mando.

El diseño y la implementación puede tener varios problemas que deben ser evitados o gestionados, algunos son.

4.9.4.12.1 La falta de una estrategia organizacional

La estrategia es la capitalización de la visión de la empresa. Sin embargo, muchas empresas se mofan de declaraciones de visión rimbombantes para complacer a las directivas, pero que son imposibles de alcanzar.

Hay muchas empresas que están contagiadas del Principio de Dilbert[23] y escriben declaraciones de visión como:

[23] El principio de Dilbert dice que las personas menos competentes son puestas en el lugar que hacen menos daño a la compañía: la dirección. El principio fue popularizado por Scott Adams en su libro del mismo título y es de lectura prácticamente obligatoria en escuelas de negocios de todo el mundo.

"Tendremos toda la riqueza del mundo mientras que todos los demás se quedan tirados en la cuneta deseando estar en nuestro lugar" [25]

Si no lo puedes medir, no lo puedes gestionar, repiten Kaplan y Norton en sus obras. Una estrategia debe poder llevarse a acciones que sean cuantificables.

Si este es el caso de la organización a la que se va a gestionar el área de TI, hay que tener claro cuál es el negocio de la organización y plantear objetivos que faciliten el modelo de negocio y la cadena de valor. Y no se debe olvidar los principios de utilidad y productividad de la tecnología: todo software hace algo útil. El fin de la tecnología es ser más productivos. A partir de estas premisas puede construirse una estrategia de TI que sea útil a la organización.

4.9.4.12.2 Falta de recursos

El área de TI requiere recursos humanos y tecnológicos. Por ello es importante vincular a las directivas y convencerles de la utilidad de la herramienta.

La implementación de un CMI de TI puede plantearse como un proyecto. De esa forma se pueden planear los recursos necesarios y saber de antemano si se van a tener. De lo contrario es mejor no iniciar una tarea para la que no se van a contar con recursos.

En el proceso de planeación es posible darse cuenta si los recursos actuales son suficientes para la gestión, por lo que es posible que con una reorganización del área se pueda implementar este modelo de gestión.

4.9.4.12.3 La tendencia a castigar los indicadores malos

El CMI es una herramienta de gestión. Se puede caer en la tentación de castigar de forma individual o grupal debido a los resultados de los indicadores del CMI, pero esto destruye el propósito del mismo. Si se hace esto se corre el riesgo de que los colaboradores solo trabajen por llenar la cifra y no por los objetivos de fondo.

4.9.4.12.4 Indicadores insuficientes o no adecuados

La gestión del área con un CMI irá informando si los indicadores faltan, sobran o están mal planteados. De acuerdo con el uso que se les dé. Un indicador que nunca se usa o que siempre está por encima de la meta es inútil.

El CMI debe ser revisado a fondo al menos una vez al año. En el ejemplo dado hay indicadores no definidos para algunos objetivos. Esto se puede dar por falta de datos, o que no se va a realizar la medición aún. U otras decisiones administrativas.

4.9.4.12.5 Olvidar los indicadores de proceso

Los procesos tienen sus propios indicadores que los llevan por el camino de la mejora continua. De forma que sean evaluados y se modifiquen/eliminen/creen.

Con la implementación de un CMI se pueden olvidar los indicadores de proceso lo cual es un error ya que siempre se puede mejorar. No se deben dejar de recolectar, evaluar y ajustar los indicadores de proceso.

4.9.5 Resumen del gobierno del mantenimiento del software

Se parte de la estrategia organizacional para llegar a una estrategia de TI con objetivos medibles y alcanzables.

A partir de estos objetivos se construyen indicadores, se lleva cada objetivo a uno o varios números que informen que tan cerca se está de la meta. Estos indicadores se enmarcan en una de cuatro perspectivas: orientación al usuario, excelencia operacional, orientación futura y contribución al negocio. Y se crea de forma gráfica un cuadro de mando integral – CMI que muestre estos indicadores en una sola hoja.

Los indicadores son de causa efecto; de resultado; inductores de actuación y vinculados a las finanzas. Por lo que deben evaluarse con frecuencia para que el director de TI tome las acciones que estime necesarias para gobernar con eficiencia el mantenimiento de software.

Este gobierno se lleva a cabo con cuatro actividades:

1. Comunicar la estrategia e instrucciones.
2. Asignar recursos.
3. Definir objetivos e instrucciones individuales de equipo y de área.
4. Proporcionar retroalimentación.

Esto hace el CMI una herramienta de gestión dinámica. Que puede o no cambiar según los resultados mostrados por los indicadores. Además, se impulsa la gestión hacia la consecución de objetivos optimizando los siempre escasos recursos.

5 Conclusiones

5.1 Por qué se debe organizar el gobierno del software en mantenimiento.

La primera conclusión es que toda empresa debe organizar la fase de mantenimiento del software de modo que optimice los recursos y apoye el negocio al que está vinculado. Esto se define cómo gobierno.

Gobernar, lo define la Real Academia Española como guiar y dirigir[24] y también enseña que su origen es la de pilotar una nave[25]. El gobierno consiste en tener la capacidad de llevar hacia un destino.

Las naves actuales: aviones, barcos y hasta los autos, cuentan con un tablero de control. Una serie de instrumentos que indican el estado en que se encuentra la nave, y que tan bien se lleva hacia el destino. Con esa información, el conductor, capitán o tripulación toma los correctivos necesarios en caso de un desvío. Para cualquiera, es lógica la existencia de este tablero.

A pesar de ello, muchos directores de tecnologías de información, gerentes de área informática, jefes de ingeniería de software, o como quiera que se llame al encargado de TI. Navegan sin tener un tablero de instrumentos que le informe del estado del software en operación. Pasan el día a día en reuniones interminables y resolviendo lo urgente sin que esto implique llevar a un destino.

Aun así, cientos de organizaciones humanas dependen de la ejecución de uno o varios programas de computadora, que fallan, necesitan cambios y operan en alguna parte de la organización. Esto necesita gestión. Necesita gobierno, que optimice el uso de los escasos recursos de la mejor manera y lleve hacia un destino: la estrategia organizacional.

Manejarlo en modo apaga incendios, o sea resolviendo el día a día, implementar un modelo de procesos o usar una herramienta no es eficiente para un software que se encuentra en operación.

5.2 La metodología propuesta para la organización de la fase de mantenimiento.

Como segunda conclusión se llega a una metodología para la organización del mantenimiento. El software o conjunto de él debe ayudar al negocio e impulsar el alcance de la estrategia. Para ello, se define una forma de trabajo, organizando áreas con marcos probados para su gestión.

No es suficiente con organizar el grupo de desarrollo, es necesario también controlar la operación diaria, La mesa de servicios que atiende el soporte, un grupo de gestión de calidad y controlar los activos son indispensables para la gestión del mantenimiento. Un

[24] Es la tercera de nueve definiciones.
[25] Del latín gubernāre

robusto sistema de gestión de la configuración ayuda a mantener el orden de los cambios. Y la gestión del conocimiento apoya en la pérdida del personal. A lo largo de esta memoria se plantean los pasos para la implementación de estas áreas de proceso.

5.3 La gestión con un cuadro de mando integral CMI

Como tercera conclusión se crea un cuadro de mando integral, que para cada organización será la clave a la hora de optimizar el uso de los recursos disponibles para alcanzar los objetivos estratégicos.

El cuadro de mando integral lleva la estrategia a objetivos alcanzables. Y los objetivos a indicadores. Estos indicadores puestos en una página son el sistema de gestión que da al jefe de TI información acerca de cómo se encuentra el software en un momento dado. Y le da datos para tomar los correctivos necesarios en caso de desvío.

El CMI es un sistema de gestión estratégica, no de evaluación del personal. E implica la optimización de recursos, siempre escasos, hacia la estrategia, tratando de resolver la paradoja de la productividad y guiando la gestión hacia la consecución de resultados.

La elaboración de un CMI no está libre de obstáculos. El primero y quizás más difícil de resolver es la falta de un modelo de negocio. Muchas de nuestras organizaciones se dedican a lo que les brinde el mercado. Y no tienen claro que hacen, quienes son sus clientes/usuarios y cuáles son sus productos/servicios. A una organización de este tipo no es posible aplicar una gestión con un CMI.

Otro de los obstáculos difíciles de flanquear es la existencia de una estrategia inalcanzable o ambigua. La estrategia se deriva de la declaración de visión y hay muchas organizaciones que tienen visiones rimbombantes que es muy difícil, por no decir imposible, de traducir a objetivos medibles. Si este es el caso y se cuenta con un modelo de negocio claro, se pueden aplicar los axiomas del software definidos en esta memoria para especificar una estrategia de TI. Estos axiomas son:

Todo software hace algo útil.
El objetivo del software es hacernos más productivos.

Una estrategia dirigida a mejorar/automatizar procesos del modelo de negocio y que lo hagan más productivo, es una guía hacia una gestión estratégica del software en operación. Con estos dos axiomas se pueden definir objetivos medibles.

La falta de recursos es otro obstáculo. Si no se tiene un compromiso de la dirección para aportar los recursos humanos, financieros y tecnológicos para la operación de TI, es muy difícil aplicar este tipo de gestión estratégica. Los indicadores de proceso, y los resultados del pasado son el insumo para determinar la cantidad de recursos que se requieren para el gobierno estratégico.

Una vez implementado el CMI se convierte en una herramienta dinámica, en un ente vivo que necesita atención. Y esta es la gestión que debe realizar el responsable de TI. Comunicar la estrategia e instrucciones. Asignar recursos. Definir objetivos e instrucciones individuales, de equipo, área y proporcionar retroalimentación. Son la gestión que llevará a tener un gobierno de TI con éxito.

5.4 Recomendaciones a los profesionales de tecnologías de la información

El software está en todas partes y cada día absorbe más aspectos de nuestras vidas. Todo software es hecho con un propósito y toda organización humana tiene una razón social de existencia. Cada profesional de tecnologías de la información debe tener claro, el objeto tanto del software como de la organización a la que está vinculado. Y contribuir desde cualquier puesto que tenga, a enriquecer ese objetivo.

La construcción de software es, al menos hasta ahora, una actividad intensamente humana. Cada día se diseñan, construyen y modifican soluciones tecnológicas para mejorar alguna tarea en la sociedad que hemos construido. Es responsabilidad de los que creamos estas soluciones, hacerlo de la mejor manera que nos sea posible.

6 Líneas de investigación Futuras

6.1 Los verdaderos costos y beneficios de las tecnologías de información

Partiendo de la paradoja de la productividad, y que una de sus explicaciones es que las inversiones en tecnología se recuperan en mucho tiempo, o que se anulan con los beneficios. Se hace necesario contar con métodos que cuantifiquen las inversiones hechas en tecnología y sus beneficios. Por ser una actividad intensamente humana, hay muchos beneficios difíciles de cuantificar. Cuantas personas se benefician, por ejemplo, de las transacciones financieras hechas por el teléfono. ¿Cómo darle un valor objetivo a esto? ¿Cómo cuantificar el trabajo hecho por personas que se auto atienden en un banco? ¿Qué obtienen citas médicas por internet? Esta es una interesante línea de investigación: los verdaderos costos y beneficios de las tecnologías de información.

6.2 El valor para la empresa del conocimiento de su personal de tecnologías de información

Otra investigación se refiere a cómo cuantificar el conocimiento de un equipo de trabajo. Se sabe, que entre más experiencia tiene una persona en un grupo de trabajo de software, mejor es su desempeño porque conoce mejor la operación, el software y los procedimientos.

A pesar de ello, las finanzas de la empresa consideran a las personas un gasto, lo que hace que los planes de compensación se orienten con los vaivenes del estado financiero de la compañía. Esto causa que, en caso de un revés financiero, una organización pueda perder el conocimiento que tienen las personas de su equipo. También, si no se cuenta con un plan de compensación adecuado, la rotación de personal puede ser muy alta, agravado con la globalización de la tecnología. Hoy cualquier profesional de TI que se lo proponga puede emplearse en prácticamente cualquier lugar del planeta. Tener una forma de llevar este conocimiento a un valor monetario, es una forma que los directivos empresariales pueden entender.

6.3 ¿Que pueden hacer los profesionales de TI para cumplir la ley?

Por último y que se sale un poco de la línea de esta memoria, se requiere una investigación acerca de la ética profesional de los profesionales de TI. En el ejercicio de la construcción de software se puede, sin saberlo o por presiones laborales, construir software que viole la ley. ¿Cuáles son las opciones para un profesional? ¿Qué hacen los colegios y asociaciones de profesionales? Si alguien se niega a construir algo porque sabe que es ilegal. ¿Cuáles son sus opciones para proteger sus derechos laborales? ¿O para salvarse de la sanción penal?

7 Bibliografía

[1] S. E. Institute, CMMI-DEV, V1.3, CMMI® para Desarrollo, Versión 1.3, Pittsburgh, Pensilvania: Software Engineering Institute, 2010.

[2] F. J. Pino Correa, Modelo de madurez de ingeniería del software, Madrid: AENOR, 2014.

[3] S. R. Huercano, Itil V3 Manual Integro, Sevilla: Biable, 2014.

[4] J. A. Peña, «Cobit 5,» Isaca, Mexico DF, 2015.

[5] Project Management Institute, PMBOK 5 edicion, Pensilvania: PMI Book service center, 2012.

[6] R. M. Solow, «We'd Better Watch Out,» *The New York Times ,* p. 36, 12 julio 1987.

[7] E. Brynjolfsson, «The Productivity Paradox of Information Technology: Review and Assessment.,» *Communications of the ACM,* vol. 36, nº 12, pp. 67-77, 1993.

[8] M. J. Vera Contreras, Modelo para evaluar la gestión de sistemas de información en las entidades publicas colombianas, Bogotá: Universidad Nacional de Colombia, 2013.

[9] M. Porter, Competitive Advantage: Creating and Sustaining Superior Performance, New York: The Free Press, 1985.

[10] E. Suarez, Plan de Viabilidad de empresa: Hermes Centro Comercial Virtual, Madrid: Escuela Europea de Negocios, 2013.

[11] S. R. Huercano, Itil V3 Manual íntegro, Sevilla: B-able, 2013.

[12] IEEE, IEEE 828 - 2012 Standard for configuration management in system and software enginering, IEEE, 2012.

[13] S. P. Berczuk y B. Appleton., Software Configuration Management Patterns: Effective Teamwork, Practical Integration., Addison-Wesley Professional, 2002.

[14] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software., Addison-Wesley Professional, 1994.

[15] K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland y D. Thomas, «Agile Manifesto,» 2001. [En línea]. Available: http://agilemanifesto.org/iso/es/manifesto.html. [Último acceso: 15 10 2016].

[16] R. S. Pressman, Ingeniería del Software, un enfoque practico-Septima edicion, Mexico: McGraw-Hill, 2010.

[17] J. Horch, Practical Guide to Software Quality Management, 2a. ed, Artech House, 2003.

[18] M. S. Merkow y L. Raghavan, SECURE AND RESILENT SOFTWARE DEVELOPMENT, CRC Press, 2010.

[19] K. Tsipenyukc, B. Chess y G. McGraw, «Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors,» *Journal IEEE Security and Privacy,* vol. 3, nº 159, pp. 81-84, 2005.

[20] D. P. N. Robert S Kaplan, El Cuadro de mando integral tercera edicion, Bogotá: Planeta, 2016.

[21] V. Grembergen y V. Bruggen, «Measuring and improving corporate information technology through the balanced scorecard technique,» de *Procedings of the Fourth European Conference on the evaluation of information tecnology*, Berlín, 1997.

[22] S. D. H. Win Van Gremberhem, Enterprise governance of Information technology, New York: Springer, 2010.

[23] P. y. G. A. L. PEREZ LORENCES, «La construcción de un cuadro de mando integral de tecnologías de la información en una empresa,» *Visión de futuro,* vol. 18, nº 2, 2014.

[24] S. Adams, El principio de Dilbert. Un autentico repaso a jefes, reuniones inútiles, manías de gerente y demas achaques laborales, Barcelona: Ediciones Juan Granica, 1997.

[25] «Web and macros,» [En línea]. Available: www.webandmacros.com/descarga/cuadro-de-mando-excel.xls. [Último acceso: 13 06 2017].

8 Glosario

A

Amazon: Es una compañía estadounidense de comercio electrónico y servicios de computación en la nube a todos los niveles con sede en la ciudad estadounidense de Seattle, Estado de Washington · 20

Apache: Software de codigo abierto de servicios web. Se usa para poner en linea sitios web · 95

Apple: Es una empresa multinacional estadounidense que diseña y produce equipos electrónicos, software y servicios en línea, con sede en Cupertino (California, Estados Unidos) y la sede europea en la ciudad de Cork (Irlanda).3 Sus productos de hardware incluyen el teléfono inteligente iPhone, la tableta iPad, el ordenador personal Mac, el reproductor de medios portátil iPod, el reloj inteligente Apple Watch y el reproductor de medios digitales Apple TV. · 32

B

Borland: Borland Software Corporation (anteriormente Borland International, Inc.) es una compañía de software, ubicada en Austin, Texas, Estados Unidos. · 19

branch: Rama en inglés. Es una copia del trunk o del directorio principal donde se realizan las modificaciones al software · 43

C

CMDB: Base de datos de configuración; herrammienta que almacena los nombre propiedades y relaciones de los elementos de configuración · 83, 84, 85, 86

CMI: Sigla de cuadro de mando integral. Es un conjunto de indicadores que muestran el estado de la empresa · 16, 22, 98, 99, 100, 101, 102, 108, 109, 110, 111, 112, 113, 114, 115

CMMI: Integración de modelos de madurez de capacidades o Capability Maturity Model Integration (CMMI siglas en inglés) es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software. · 12, 16, 17, 36, 40, 65

CMS: Sistema de gestión de contenido. Software para la publicacion de sitios web como blogs, tiendas en linea o catalogos de servicios · 95, 96

COBIT: Control Objectives for Information and related Technology,Objetivos de Control para Información y Tecnologías Relacionadas es una guía de mejores prácticas presentada como framework, dirigida al control y supervisión de tecnología de la información (TI). Mantenida por ISACA (en inglés · 18

CRC: Las tarjetas CRC son una herramienta usada como metodología para el diseño de software orientado a objetos, creada por Kent Beck y Ward Cunningham. Consiste en dibujar una tarjeta por cada clase u objeto, y dividirla en tres zonas; el nombre de la clase. Las responsabilidades de la clase. Son sus objetivos, a alto nivel. Las responsabilidades, los colaboradores, que son otras clases que ayudan a conseguir cumplir a esta con sus responsabilidades · 63

D

Drupal: CMS libre, modular, multipropósito y muy configurable que permite publicar artículos, imágenes, archivos y que también ofrece la posibilidad de otros servicios añadidos como foros, encuestas, votaciones, blogs y administración de usuarios y permisos · 95

G

I

J

M

O

P

PHP: Lenguaje de programación interpretado de uso amplio en Internet. Gran parte de los sitios web estan escritos en este lenguaje · 95

PMO: Oficina de gestion de proyectos, por el ingles project management office. · 27

PMP: Project Management Professional (PMP) es una certificación (credencial) ofrecida por el Project Management Institute (PMI). · 12, 16, 19

T

Tag: Etiqueta en inglés. Directorio copia de un sistema de control de versiones. Suele ser una version del sofware · 43

TI: Acrónimo de Tecnología de la Información. Se refiere al hardware, software, servicios de comunicaciones y todo lo relacionado a informática · 12, 13, 15, 16, 17, 19, 20, 21, 22, 26, 27, 29, 31, 33, 94, 96, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 117

Trunk: Tronco en ingles. Es el directorio principal del sistema de control de versiones implementado en la mayoría de aplicaciones libres y comerciales · 43

U

UML: Lenguaje de modelado unificado. Es un estandar para realizar diagramas de un software · 28, 30, 31, 39

W

Wordpress: CMS escrito en PHP de amplio uso en la red (véase CMS) · 95