Proceso de la ingeniería del software

TEMA 1 GESTIÓN DE LA CALIDAD DEL SOFTWARE

Tema 1
Gestión de la calidad del software

1. QUÉ SIGNIFICA TRABAJAR CON CALIDAD

Calidad del software es concordancia con los requisitos funcionales (requisitos del cliente) y de rendimiento establecidos, con los estándares de desarrollo documentados de manera explícita y con las características que se espera de todo software desarrollado profesionalmente.

Es decir, los aspectos de calidad los podemos clasificar en dos aspectos bien diferenciados:

- Un primer grupo que corresponde a los aspectos en los que se fijaría un usuario experto: cubre sus necesidades funcionales, funciona correctamente y sin errores, se entrega en el tiempo previsto, es fácil de utilizar, dispone de un buen manual de usuario, tiene un buen servicio de mantenimiento y actualizaciones.
- Un segundo grupo en los que se fijaría un experto informático: Su diseño y arquitectura, bases del desarrollo son sólidos y correctos, los requisitos y arquitectura están bien documentados, sigue un estándar de codificación bien pensado, el código tiene los comentarios necesarios, las versiones disponen de un buen control de configuración.

Suele tenderse a identificar la calidad simplemente con la presencio o no de errores de funcionamiento y olvidar los demás aspectos; esto hace que muchos identifiquen plan de calidad (mejora continua) con plan de verificación (testeo de funcionamiento), cuando realmente el testeo es solo un punto del plan de calidad.

2. LA MEJORA DE PROCESOS

Bien, siendo realistas diremos que son muy pocas veces las que empezamos un proyecto desde cero y tenemos las manos libres para hacer las cosas de la mejor manera posible o como a nosotros nos gustaría. Lo más frecuente es tener muchas restricciones que nos limitan las opciones de desarrollo del proyecto: porque estamos en una fase de mantenimiento y hemos heredado un código que no nos permite ciertos cambios, porque la fecha de entrada ya está fijada y no hay tiempo, porque los recursos disponibles en tiempo y/o número y/o formación son pequeños…

La clave para mejorar por tanto, reside en ir introduciendo mejoras poco a poco. Las **micromejoras** por insignificantes que parezcan son la base para la puesta en marcha de un proyecto de mejora de procesos; siempre será mejor ir poco a poco que no hacer nada y aunque vayamos despacio no por ello debemos renunciar a iniciar un camino hacia la mejora del proceso global estandarizado en la empresa, que es nuestra finalidad.

Hay que comenzar estas micromejoras con prudencia, en un solo proyecto y en un solo equipo de trabajo, escogiendo el proyecto y equipo que por circunstancias resulte más adecuado. Dentro de este proyecto elegiremos un único proceso, normalmente en el que ya hace tiempo que queríamos introducir una mejora y todavía no se ha realizado o lo queremos mejorar porque es en el que solemos tener más problemas.

Realizada la micromejora obtenemos información sobre los beneficios que nos aporta y las dificultades que se nos han planteado en su implantación. Con

toda esta información y experiencia, ya podemos pasar a implementar la misma mejora en otro proyecto. La aplicación en esta segunda fase será más sencilla gracias a la experiencia previa y serán las mismas personas que la han llevado a cabo las que la presenten y lo expliquen al nuevo equipo de trabajo.

El paso siguiente es comparar los resultados entre los distintos proyectos y recopilar una experiencia que nos ayudará a evaluar, de acuerdo con los resultado sy los objetivos perseguidos, cómo lo estamos llevando a cabo.

La **comunicación** es fundamental para llegar a buen puerto. Tenemos que explicar, clarificar y justificar por qué lo hacemos y cómo lo hacemos; y hemos de comunicárselo a los jefes, a los usuarios de la aplicación, al equipo de desarrollo; y debemos comunicarnos de manera fluida, escueta, concisa y continuada.

En general, las mejoras no darán un fruto palpable o medible enseguida y puede pasar que en un primer momento sólo se aprecia un coste de tiempo adicional y que no se valore a largo plazo lo que esta inversión temporal inicial puede ahorrar en tiempo y dinero; ello hace que la comunicación sea fundamental para intentar despejar estas dudas.

3. NORMATIVA DE LA CALIDAD

Son varios los organismos que ya hace tiempo que se preocupan por desarrollar unos estándares que ayuden a las organizaciones dedicadas al desarrollo de software a seguir unas pautas que conduzcan a garantizar un producto de calidad. Todos se basan en el argumento de que, si el proceso de desarrollo de un producto es de calidad, el producto resultante también lo será.

Los estándares más frecuentes son SPICE y CMM.

SPICE: ISO 15504

El objetivo de SPICE (Software Process Improvement and Capability dEtermination) es desarrollar un estándar internacional para la consultoría en procesos de software que permita elaborar un plan de mejora de procesos, que defina muy bien los riesgo sy las prioridades con el fin de determinar el nivel de madurez de los procesos que componen el desarrollo de proyectos de software en una organización o empresa. La manera de trabajar de spice consiste en:

- Desarrollar un borrador de trabajo de un estándar para la consultoría en processo de software y la determinación del nivel de madurez de un proceso.
-
- Llevar a cabo pruebas piloto en el ámbito internacional basadas en este estándar.
- Promover por todo el mundo la transferencia de conocimientos en procesos de software

Niveles de madurez de un proceso según SPICE

Nivel 0. Incompleto
- El proceso se lleva a cabo de manera caótica e incompleta.

Nivel 1. Informal
- El proceso se lleva a cabo de manera intuitiva, se dispone de las entradas y salidas del proceso.

Nivel 2. Planificado y seguido
- Hay una gestión del proceso y un responsable del mismo.

Nivel 3. Bien definido
- Hay un proceso básico definido que se adapta a las necesidades específicas de cada proyecto.

Nivel 4. Controlado cualitativamente
- Hay un registro de métricas que permite el control objetivo de los resultados del proceso.

Nivel 5. Mejora continuada
- Los resultados del análisis de las métricas permiten introducir mejoras continuadas en los procesos.

El estándar CMM

El objetivo del CMM es ayudar a las organizaciones de software a mejorar el nivel de madurez de sus procesos para que sigan un camino que evolucione desde el proceso caótico al proceso organizado. Este estándar se gestiona y coordina dentro del Software Engineering Institute (SEI) y al contrario que SPICE que basa todo su análisis en los procesos individuales, el CMM se basa en el conjunto de la organización del software. El CMM clasifica los niveles tal y como podemos ver en la tabla lateral.

Niveles según CMM

Nivel 1. Inicial
- El proceso de desarrollo de software es caótico; hay pocos procesos definidos y el éxito radica en la capacidad y esfuerzo individual.

Nivel 2. Repetible
- Se establecen algunos procesos básicos de gestión para hacer un seguimiento de costes, calendario y funcionalidad.

Nivel 3. Definido
- El proceso está documentado, estandarizado e integrado en los procesos globales de la compañía.

Nivel 4. Gestionado
- Hay una recopilación de medidas detallada sobre el proceso de desarrollo de software y la calidad del producto.

Nivel 5. Optimizado
- Hay un proceso de puesta en marcha de mejoras de manera continuada basado en los resultados cuantitativos del proceso y en las pruebas piloto.

4. PUESTA EN MARCHA DE LA MEJORA DE PROCESOS

Dos aspectos son fundamentales para la puesta en marcha de una buena mejora de procesos: analizar el punto de partida y definir una estrategia.

En el **análisis del punto de partida** son varios aspectos los que tendremos en cuenta:

- Producto: No es lo mismo un software de control de stocks en un supermercado, que un software de control del piloto automático de un avión; el nivel de criticismo es distinto y por lo tanto, el método de trabajo y plan de calidad para cada uno de ellos también lo será.
- Usuarios: Si disponemos de usuarios expertos que aportan ayudas y sugerencias es bueno para el proyecto, aunque pueden tratarse también de usuarios poco formados y poco reticentes al cambio; cada caso nos conducirá a trabajar de una manera distinta.
- Mercado/Competencia: hay que tener presente qué hacen los demás, y saber si queremos dar una imagen de software puntero y rápido, que respetamos las fechas de entrega, calidad sin ningún tipo de error…
- Tamaño del proyecto: Depende del tamaño del equipo de trabajo y su perfil, aunque también de la ubicación del mismo (incluso en países distintos simultáneamente).
- Historia del proyecto/producto: No es muy frecuente partir de cero en un desarrollo; la gran mayoría de las veces hay que hacer modificaciones o ampliaciones a productos ya existentes.

La **definición de una estrategia** nos lleva a su vez, a analizar varios puntos:

- Estrategia de la compañía: Depende si queremos desarrollar un programa de uso interno, de uso interno para luego comercializar, desarrollar con tecnología punta, o con tecnología más madura libre de errores.
- Recursos: ¿Cuanto dinero, tiempo y equipo puedo invertir? Hay que tener en cuenta que la empresa emprende el proyecto para obtener unos beneficios; dependiendo de lo que esperen obtener así se invertirá.

Analizado lo anterior procedemos a identificar los procesos más relevantes en el desarrollo de nuestro software (recogida de requisitos, diseño, evaluación, arquitectura, implementación…) y luego pasamos a identificar las carencias (menos equipo del deseado), riesgos (tiempos de entrega demasiado ajustados), etc. Al final tendremos que ser capaces de traducir cada uno de estos problemas a pequeños planes de mejora, micromejoras que además sean realistas y se puedan llevar a cabo.

Es fácil que en la puesta en marcha de estas micromejoras, encontremos reticencias por parte de los equipos de trabajo, falta de tiempo, diversidad de opiniones… Para salvar estos obstáculos las pequeñas mejoras en los procesos tienen que salir de manera natural, como propuesta del propio equipo de desarrollo. Si las mejoras se deciden entre todos durante una lluvia de ideas bien estructurada y dirigida, y las propuestas salen del mismo equipo de trabajo, será más fácil que todo el mundo ponga lo mejor de sí mismo para sacar adelante el proyecto.

Una vez identificadas las propuestas, tenemos que establecer prioridades. No lo podemos hacer todo a la vez, hay que empezar por las más urgentes y fáciles de implementar, y clasificar las micromejoras según sean a corto, medio o largo plazo. Es conveniente que el equipo de desarrollo también participe en la discusión de las prioridades, ya que será la manera de que todo el mundo entienda las razones de éstas.

Con toda esta información obtendremos un **plan de trabajo,** bien definido para poner en marcha el proyecto de mejora de procesos.

5. PLAN DE ACCIÓN

Cuando estemos en disposición de hacer el **plan de calidad** de un proyecto, hay que recordar que debe incluir acciones para todo el ciclo de vida del proyecto y debe ser aprobado en el momento en que se aprueba el lanzamiento del proyecto; debe incluir:

- Plan de proyecto.
- Control de configuración
- Control de cambios
- Análisis y gestión de riesgos
- Plan de verificación funcional
- Plan de calidad en la construcción
- Auditoría externa

Los tres primeros puntos corresponden a tares específicas en la gestión de proyectos, por lo que redundaremos en las 4 últimas.

Análisis y gestión de riesgos

Como el tiempo y los recursos son limitados, no podemos verificar el 100% del funcionamiento y calidad del software ¿por donde empezamos y qué dejamos de verificar? Tenemos que empezar por las áreas de desarrollo que tienen un riesgo más elevado y, por lo tanto, son más críticas, a continuación debemos ser capaces de llevar a cabo la gestión de estos riesgos de manera que los tengamos bien numerados, descritos y ordenador.

Debemos hacer dos tipos de verificaciones:

Plan de verificación funcional

También denominado black box y es todo lo que se verifica sin saber qué hay dentro del programa; es todo lo que podría verificar un usuario de la aplicación sin conocimientos de informática y sus objetivos son asegurar que el producto implementa la funcionalidad pedida, que el programa funciona correctamente sin errores, que la aplicación es suficientemente rápida y fácil de utilizar y, en general, que el cliente y/o usuario se lleva una buena impresión. Esta verificación la pueden hacer los propios usuarios o grupos de usuarios, pero no como usuarios, sino como participantes en el equipo de desarrollo. Las acciones que se pueden realizar en la verificación funcional son las siguientes:

- Revisión de los requisitos de usuario: Ver que son completos, correctos, consistentes y definidos; en caso de que se hagan prototipos, el equipo de verificación funcional será el encargado de valorar el prototipo y darle el visto bueno.
- Determinación del plan de verificación funcional: Tiene que permitir transmitir información cuantitativa sobre el nivel de calidad, estado del proyecto y, por lo tanto, nivel de riesgo que se asume Este ejercicio todavía nos dará más información cuando se haya llevado a cabo varias veces, porque conoceremos los resultados de las experiencias anteriores y podremos compararlas.
- Casos de prueba y su gestión: son un listado de acciones que, de manera ordenada y organizada, permite ir probando todos y cada uno de los casos con los que el programa se podrá encontrar cuando se ejecute en el entorno real; contará con características como test de rendimiento y velocidad, de usabilidad, test de carga, de seguridad, test de instalación y test de volumen de datos. La ejecución debe ser metódica, anotando en cada caso qué prueba se realizó, en qué condiciones y qué incidencias se produjeron.
- Congelación del código: Una vez sepamos cual es la fecha concreta de la entrega de la aplicación, es necesario definir una fecha anterior (dependerá del tamaño del programa) a partir de la cual ya no se pueden introducir funcionalidades nuevas, mejoras o cambios en el programa, sólo estará permitido tocar el programa para arreglar errores. En caso de que durante este intervalo de tiempo sea imperativo modificar el código para introducir mejoras o cambios, se tendrá que aprobar explícitamente y todo el mundo tendrá que aceptar que existe el riesgo de introducir errores.
- Aplicaciones de Internet: Todo lo dicho anteriormente es igualmente válido para aplicaciones en Internet, pero hay que añadir además pruebas específicas dado que hay menos herramientas para verificar este tipo de aplicaciones y la

complejidad de las mismas. Habrá que realizar test de compatibilidad (entre equipos, sistemas operativos y navegadores), test de navegación, de interacción, de carga.

Plan de calidad en la construcción

El objetivo principal de la verificación en la construcción es asegurar la calidad de todo aquello que no se ve en el producto de software y es todo lo que se verifica desde un punto de vista técnico en las entrañas del código. Esta verificación la tiene que llevar a cabo un informático experto y es lo que se suele llamar white box.

Con la verificación en la construcción queremos conseguir garantizar que la aplicación en general estará bien construida por dentro, es decir, será fácil de mantener, robusta y flexible ante cambios de requisitos. Las acciones que incluye esta verificación son: revisar la arquitectura y el diseño, certificar un proceso en la construcción, constatar unos estándares de codificación, revisión del código, test de integración periódico…

Tanto el plan de verificación funcional como el de la construcción nos tienen que permitir obtener datos cuantitativos, objetivos y comparables que nos darán información sobre la evolución de nuestra manera de trabajar y nos aportarán conclusiones que nos servirán para la mejora de procesos.

Auditorías externas

La calidad de los elementos con los que hemos verificado la calidad en la construcción como la verificación funcional, pueden condicionar la calidad final de nuestro producto. Por ello es importante comprobar la solidez de la empresa que ha desarrollado estas herramientas de verificación, contactar con otras empresas que las utilicen para saber su grado de satisfacción y contactar con la empresa fabricante para conocer sus planes de desarrollo futuros.

TEMA 2 GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE

1. CONCEPTOS BÁSICOS

La gestión de la configuración del software (GCS) es el proceso que identifica y define los componentes de un sistema, controla las distribuciones de los productos y cambios que se producen durante su ciclo de vida, registra e informa del estado de los componentes y sus peticiones de cambio y verifica que los componentes son completos y correctos respecto de las especificaciones.

De la definición podemos deducir que los elementos principales de la GCS son la identificación, el control de cambios, el control de estado y las auditorías.

Con una buena gestión se busca poder conocer en cualquier momento el estado de cada uno de los componentes de un sistema, cómo se ha llegado a éstos, por qué han sido modificados y quién lo ha hecho.

Muchos jefes de proyecto y desarrolladores no perciben un gran valor añadido en la GCS, la entienden como una sobrecarga de trabajo y un exceso de burocracia; pero problemas como reparar un mismo error de software más de dos veces o no saber qué versión del código fuente es la que hay que reprogramar por no saber cual está en producción en esos momentos, son errores frecuentes con fácil solución mediante una buena GCS.

Entre los conceptos básicos destacan:

- Elemento de configuración: es un producto generado durante el desarrollo de un proyecto o una pieza de código que se trata como una única entidad desde el punto de vista de la gestión de la configuración. Dentro de ellos hay elementos básicos o componentes (un fichero del manual de usuario, por ejemplo) y elementos agregados (todo el sistema, o un subsistema más o menos independiente del mismo).
- Configuración de un proyecto es el conjunto de características funcionales y físicas del software, descritas en la documentación e implementadas en los programas que se quieren controlar.
- Una versión identifica el estado de un elemento de configuración en un momento definido del tiempo; distintas versiones de un mismo elemento se diferencian por la corrección de un error, un cambio de funcionalidad, etc.
- El control de versiones es el procedimiento para identificar y gestionar los elementos de configuración a medida que cambian con el tiempo, normalmente con una herramienta especializada.
- La línea base es un conjunto de elementos de configuración que han sido revisados y aprobados formalmente por la dirección y/o el cliente y que sirve de base para el desarrollo posterior del proyecto.
- Una petición de cambio es una solicitud formal para cambiar uno o más elementos de configuración de la línea base; se deben tratar formalmente lo que implica que hay que analizarlas, estudiar su impacto en el proyecto y, en su caso, aceptarlas, documentando las razones para la toma de cada decisión.
- Una promoción es una versión que se pone a disposición de los otros participantes en el proyecto.
- Una distribución es una versión que se pone a distribución del cliente final.
- Una librería es el espacio físico en el que se almacenan los elementos de configuración elementales con el conjunto de funcionalidades que permiten nombrar e identificar las distintas versiones de los elementos de configuración y controlar el estado de los cambios en estos elementos.

2. ACTIVIDADES DE LA GESTIÓN DE LA CONFIGURACIÓN

Varias son las actividades fundamentales de la gestión de la configuración:

Identificación

Se tienen que identificar varios tipos de elementos: los básicos que formarán parte del proyecto y del sistema, la versión que se utilizará, las librerías en las que se almacenarán las distintas versiones de los elementos. Si no se tienen los elementos identificados, no se podrá hacer un seguimiento de los cambios que se incorporen y por tanto, la identificación debe comenzar al inicio del proyecto.

Los elementos básicos son como mínimo, los documentos de requisitos del proyecto y todo el código fuente que se general. Los elementos documentales incluyen los documentos de requisito, los diseños técnicos, la definición de interfaces con el usuario, los manuales de uso, etc. Hay que identificar asimismo la jerarquía de elementos del proyecto; un sistema es el resultado final del proyecto y está formado por subsistemas (por ejemplo un sistema para controlar la gestión de una tienda puede tener un subsistema de comunicaciones, otro de gestión de stocks…); un subsistema se divide en aplicaciones, etc. y esta descomposición puede tener distintos niveles hasta llegar a los elementos básicos; al inicio del proyecto no estamos en condiciones de llegar hasta estos elementos básicos en un desglose completo pero tendremos que llegar hasta el máximo nivel de detalle que podamos.

Definir un sistema de versiones implica definir la manera de identificar cada versión de un elemento y además el modo de asegurar que en cada momento se trabaja con la versión correcta y que no se modifica accidentalmente una versión inadecuada. Un sistema muy utilizado para esta identificación consiste en dos números separados por un punto; el primero indica la versión y sólo se incrementa cuando se le hacen grandes modificaciones funcionales; y el segundo indica la revisión, que se incremente cuando se hacen pequeñas mejoras o correcciones del sistema. Hay situaciones en las que es necesario hacer un desarrollo en paralelo, es decir, que dos personas o grupos diferentes modifiquen un mismo elemento al mismo tiempo; esta situación puede causar problemas, ya que es fácil perder información si no se trata con precaución. En el caso de un desarrollo en paralelo es necesario definir un procedimiento para fusionar las dos ramas de forma no traumática.

Hay puntos durante el desarrollo del proyecto en los que ciertos elementos de configuración se aprueban y se dan por finalizados; en este punto se debe congelar estos elementos y así la versión aprobada no se puede modificar bajo ningún concepto a no ser que haya una petición de cambio formal; y se incorpora ya a la línea base. Durante el ciclo de vida del proyecto, el contenido de esta línea base cambia; al principio está vacía y al final del proyecto contendrá todos los documentos y elementos del software que integran el producto.

Control de cambios

La gestión de cambios es el conjunto de tareas para controlar las modificaciones que se hacen sobre los elementos de configuración que están en la línea base, de manera que siempre se garantice la integridad entre los productos desarrollados.

Cuando se genera una petición primero hay que analizarla (generar toda la documentación posible para ese cambio), luego se evalúa el impacto en el proyecto, se registra con todo esto una resolución (aplicar el cambio o no); si se acepta el cambio puede ir a mantenimiento o a modificación; se notifica la resolución al originador del cambio y se monitoriza hasta que se incorpora a la línea base.

Parte de la gestión de la configuración incluye la definición de las librerías que se utilizarán y cómo se hará su control de acceso, tanto para extraer sus elementos como para incorporarlos. No se deben permitir la incorporación de

elementos en una librería sin una revisión previa que verifique que la incorporación es correcta y que no se creará ningún tipo de problema.

Estado de la configuración

El objetivo de controlar el estado de la configuración es proporcionar a la dirección del proyecto visibilidad del estado en el que se encuentran los elementos de configuración y, por lo tanto el estado del proyecto. Una parte importante de esta información son las métricas, que hay que definir de manera que los resultados obtenidos proporcionen información real sobre el estado del proyecto y permitan tomar unas decisiones acertadas.

Auditorías

Periódicamente se deben hacer auditorías para comprobar que se están siguiendo rigurosamente todos los procedimientos definidos para controlar la configuración y que los productos se están desarrollando según los requisitos y los estándares definidos. Las auditorías las pueden hacer tanto personas externas al equipo de desarrollo como personal interno con responsabilidades de gestión de la configuración asignadas. Hay varios tipos de auditorías:

- Funcionales: Comprueban que los productos finales (el sistema que se entregará) cumplen todos los requisitos del proyecto.
- Físicas: Comprueban que el diseño y los documentos, como por ejemplo los manuales de usuario, de mantenimiento o instalación, son consistentes.
- Internas: Son más informales y se hacen durante el desarrollo, antes de las auditorías funcionales o físicas. Su objetivo es evitar que se propaguen errores e inconsistencias, dado que detectan los problemas durante el desarrollo. Se acostumbran a hacer sobre partes del proyecto y no hay que esperar a finalizar una fase para llevarlas a cabo.
- Trazabilidad: comprueban que para cualquier producto final, se puede encontrar el requisito o requisitos que lo han originado. Esto se hace para evitar la introducción en el sistema de requisitos no documentados. Muchas veces, a causa de las prisas y las presiones, aunque haya un buen control de cambios, hay cambios que quedan indocumentados.

Al acabar el proyecto y antes de entregar el sistema al cliente, se deben hacer una auditoría funcional y otra física, para comprobar la calidad de los productos.

Manufactura

Se entiende por manufactura el proceso de crear el sistema que se debe entregar al cliente final, a partir de todos los componentes que lo integran. Dicho de otro modo, el proceso de manufactura consiste en generar una distribución que se creará a partir de la línea base que está en la librería de archivos de producción.

3. ROLES Y RESPONSABILIDADES EN LA GESTIÓN DE LA CONFIGURACIÓN

Las responsabilidades en la gestión de un proyecto se asignan a roles, aunque en un proyecto pequeño puede ser que una sola persona asuma todos los roles (o distintos a tiempo parcial). En un proyecto mediano o grande será necesaria la dedicación a tiempo completo de distintas personas que asumirán los distintos roles.

- Gestor de la configuración: Tiene la responsabilidad de llevar a cabo las actividades relacionadas con la gestión de la configuración del proyecto. Desarrolla, documenta y distribuye los procedimientos de la gestión, establece el sistema de líneas base y backup, asegura que en la línea base están incorporados los elementos necesarios y solo se hacen cambios autorizados.

Este rol es imprescindible, en proyectos pequeños puede ser asumido por el jefe de proyecto y otra persona del equipo, lo importante es que todos los que están involucrados en el proyecto sepan quién asume el rol.

- Junta de control de cambios: Grupo de personas que controlan la configuración de un proyecto. Son los responsables de evaluar, aprobar o rechazar los cambios solicitados sobre los elementos clave; deben asegurarse que realmente todos los cambios solicitados se analizan y evalúan antes de que se tome una decisión sobre su viabilidad y que todos los afectados por su posible cambio reciben una notificación de la decisión tomada antes de que se ejecute.
- Miembro del equipo de desarrollo: Son todos los miembros que participan directamente en la gestión de la configuración, utilizando los procedimientos y mecanismos definidos por el responsable de la GCS, crean y modifican los componentes del proyecto cuando es necesario.
- Auditor: Cumple el calendario concertado de auditorías, siguiendo los procedimientos definidos y asegura que los productos cumplen los estándares definidos.
- Administrador de herramientas: es la persona responsable de la administración y el mantenimiento de las herramientas utilizadas para la gestión de la configuración.
- Responsable de calidad: persona o grupo de ellas que deben definir el plan de calidad de un proyecto y asegurar su cumplimiento durante el desarrollo y mantenimiento posterior de los productos generados. Debe además comprobar si se utilizan los procedimientos y debe transmitir la información necesaria para que los responsables de los procedimientos puedan evaluar las necesidades de mejora.

4. GESTIÓN Y HERRAMIENTAS DE LA GESTIÓN DE LA CONFIGURACIÓN

Al inicio de un proyecto se escribe el plan de la gestión de la configuración, donde se documenta la manera de llevar a cabo la GCS; todos los implicados en el proyecto mientras dura su desarrollo, pueden acudir a este documento para conocer el proceso que seguirá la gestión de la configuración. Ya comentábamos en el tema anterior que hay varios estándares, de la IEEE y de la NASA, un posible índice de contenidos de un plan de GCS podría contener los siguientes puntos: Introducción, organización, actividades de la gestión de la configuración (especificación de la identificación,, mecanismos de control, registro del estado de la configuración, auditorías), formación y otros puntos específicos.

Junto con el plan de gestión de la configuración, es importante definir un esquema de base de datos para mantener actualizada toda la información referente a la configuración que nos proporcione información que ayude a hacer un buen análisis de impacto de una petición de cambio y que también proporcione información y métricas para conocer el estado de la configuración.

Actualmente existen en el mercado muchas herramientas que ofrecen algún tipo de apoyo a la gestión de la configuración. Frente a tanta oferta es importante tener algunos criterios que ayuden a evaluarlas, según las necesidades y el tipo de proyectos. Las características que este tipo de herramientas deben tener son varias:

- Robustez: Debe ser fiable (correcto funcionamiento del sistema), consistente (mantiene la información de manera coherente).
- Funcionalidad: Capacidad de definir una base de datos a medida para cada proyecto, con un buen control de versiones y de ramas y de las peticiones de cambio.
- Usabilidad: Buena interfaz de usuario, adaptable, con ayuda interactiva y documentación escrita.
- Facilidad de administración: fácil de instalar y actualizar.

- Facilidad de integración: con distintas plataformas e incluso con otros programas que ayuden a la gestión de la configuración.
- Disponibilidad comercial: calidad del servicio y penetración de la empresa en el mercado.
- Capacidad para extraer métricas.

Si se quiere tener éxito en la implementación de una buena gestión de la configuración, elegir una buena herramienta es necesario, pero no suficiente. Antes hay que estudiar las necesidades de la organización desde el punto de vista de su complejidad técnica. Hay que considerar aspectos como, por ejemplo, la cantidad de software que la herramienta ha de manejar, las plataformas sobre las cuales se quiere mecanizar, las interfaces con otras herramientas que se necesitan, etc.

TEMA 3 MANTENIMIENTO DEL SOFTWARE

1. TIPOS DE MANTENIMIENTO DEL SOFTWARE

La etapa de mantenimiento del software, además de ser la parte más olvidada del ciclo de vida de un sistema informático, mueve un gran volumen de negocio (y de dinero).

El mantenimiento del software es el conjunto de actividades que gestiona los cambios que se producen en el software informático cuando ya se ha entregado al usuario para garantizar su funcionamiento correcto y el alcance de los objetivos definidos desde el principio hasta el final de la utilización.

Este mantenimiento se puede clasificar en los siguientes tipos:

- **Mantenimiento correctivo**: Es un proceso de diagnosis y corrección, de acción y reacción: ante un error (acción) se pone en marcha el proceso de intentar solucionarlo (reacción). El mantenimiento correctivo lleva a cabo reparaciones y modifica el código necesario para dar solución a un problema. Ha que hacer un mantenimiento correctivo cuando un programa da un error de funcionamiento, cuando da resultados que no coinciden con los definidos en los requisitos y cuando la documentación del sistema no coincide con el software desarrollado. Cuando se ha llevado a cabo una corrección, las pruebas de aceptación de cambios deben ser mucho más severas que en el proceso de desarrollo. Hay a su vez dos tipos de mantenimientos correctivos:
    - **Recuperación de emergencia**: Se hace en u período muy corto de tiempo; normalmente afecta a muy pocos programas.
    - **Reparación planificada**: Soluciona problema son urgentes y también se plantea como revisión de las reparaciones de emergencia que se han llevado a cabo.

  El proceso de mantenimiento correctivo puede llevarse a cabo a través de una de estas 4 estrategias:
    - Top-Down (descendente): Cuando las causas de error no son evidente sy no se sabe por donde empezar, se sigue una estructura lógica y de navegación del programa hasta llegar a los posibles orígenes del problema.
    - Bottom-up (ascendente): Cuando se conocen los síntomas del problema y, sobre todo, su origen, se empieza a repasar la lógica del programa a partir de este punto hasta detectar el problema.
    - Debugger: Cuando fallan las anteriores, se utiliza el lenguaje de programación del sistema para analizar los cambios internos en el programa hasta encontrar el momento en que se produce el error.
    - Seguimiento de la traza: Cuando el sistema es tan complejo que no permite un seguimiento interno, se hace una traza del sistema y se analizan sus acciones y los cambios que introduce.
- **Mantenimiento adaptativo**: Se debe a la evolución de los equipos informáticos y a la aparición de nuevas herramientas y de plataformas más adecuadas para el antiguo software. Un ejemplo de este tipo de mantenimiento es la adaptación de sistemas a nuevas versiones de sistemas operativos. Para el mantenimiento adaptativo el proceso es el mismo que el del ciclo de vida clásico del software: definición de requisitos, revisión de diseño del sistema, revisión del diseño de datos, revisión de la documentación y revisión del impacto para determinar las consecuencias de los cambios. Hay 3 factores que influyen en este tipo de mantenimiento:
    - **Las necesidades internas de negocio**: Necesidades individuales de cada empresa.
    - **La competencia externa**.
    - **Cambios externos**: Por ejemplo cambios en la legislación, o la introducción de nuevos impuestos que varían el sistema de nóminas, etc.

- **Mantenimiento perfectivo**: Es el tipo de mantenimiento al que las organizaciones destinan más recursos en tiempo y dinero. Además de introducir mejoras en el sistema a petición del cliente, el mantenimiento perfectivo también corrige los errores de rendimiento y mejora los procesos de manera que sean más eficientes y fáciles de mantener. Cuando hay que incorporar nuevas funcionalidades, el proceso es el mismo que el descrito para el mantenimiento adaptativo, es decir, seguimos las etapas del ciclo de vida del software habitual.
- **Mantenimiento preventivo**: Se hace para prevenir los errores antes de que se produzcan: se trata de acciones de anticipación. Son susceptibles de este tipo de mantenimiento todos aquellos programas que hayan estado en uso durante un determinado número de años, programas que se están utilizando sin presentar errores y programas que necesitarán en un futuro próximo una serie de modificaciones o mejoras importantes.

2. CARACTERÍSTICAS DEL MANTENIMIENTO

Hay varios problemas usuales con los que nos encontramos a la hora de afrontar el mantenimiento de un sistema: Es difícil comprender el código implementado por otras personas (además no ser en absoluto atractivo para los profesionales dedicarse al mantenimiento de sistemas antiguos), la documentación del sistema no se ajusta a la realidad o el sistema no ha sido diseñado teniendo en cuenta un mantenimiento posterior.

Hay características que hacen de un sistema fácil de mantener:

- Facilidad de prueba: Facilidad con la que se pue3de demostrar la corrección de un software.
- Inteligibilidad: Facilidad de entender un software a partir del código y la documentación.
- Portabilidad: Facilidad con la que un software puede ser trasladado a un nuevo entorno informático.
- Eficacia: Capacidad del software para cubrir las necesidades de los usuarios.
- Eficiencia: Capacidad del software para hacer sus funciones sin cargar excesivamente los recursos del hardware.
- Facilidad de uso: Indica si el software elaborado es útil, fácil de utilizar y de aprender para los usuarios.

Características especialmente importantes son:

- Nomenclatura del software: El hecho de nombrar las variables y funciones con nombres significativos facilita un rápido entendimiento y seguimiento del programa. Así si una variable se llama total_gastos da más información sobre su función que si se denomina todo_g o tg.
- Indentación de los programas: Una correcta indentación facilita la lectura y el entendimiento de la lógica del programa.

Si el software estuviese bien estructurado, bien documentado con diagramas de flujo de datos, diagrama de procesos y tuviese un buen diseño de base de datos y unos estilos de programación correctos, las tareas de mantenimiento serían más fáciles y menos costosas. Puesto que estas condiciones no siempre se cumplen, el equipo de mantenimiento debe entender la estructura de datos, el flujo de datos, los procesos del sistema y las diferencias entre posibles versiones.

Estas tareas son más fáciles de entender con la ayuda de herramientas especializadas, que facilitan la visualización del modelo de datos, y dejan traza de todos los hechos relevantes que se han ejecutado, facilitando el seguimiento de procesos y la identificación de los cambios que se producen en el sistema.

Las herramientas de ayuda al mantenimiento del software van surgiendo inicialmente como herramientas de desarrollo, para mejorar su calidad y, de esta

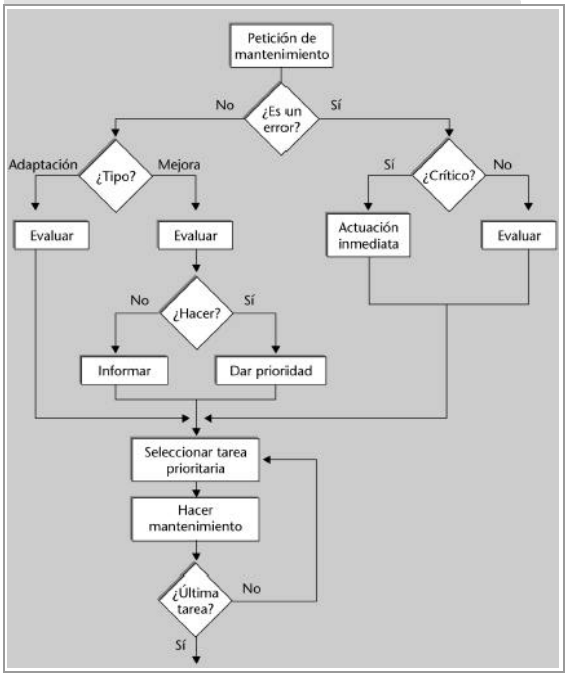
manera, reducir su mantenimiento posterior, por lo tanto, estas herramientas se utilizan en las dos fases.

Cuando se hace un nuevo software en el entorno de desarrollo (donde los programadores efectúan las tareas de creación de nuevas funcionalidades y el mantenimiento de las antiguas), después se traslada al entorno de pruebas donde se llevan a cabo las pruebas unitarias y globales, y una vez que pasa todos los test, se pasa a la explotación (entorno real) donde los usuarios trabajan con el sistema. En caso de detectarse un error o deficiencia, el programa se pasa al módulo de desarrollo y se modifica; a continuación se pasa al entorno de pruebas en el que se somete a prueba y, finalmente se traslada al entorno de explotación para que los usuarios lo puedan utilizar.

En muchas ocasiones, puede parecer más conveniente adquirir un producto informático existente en el mercado que crear uno nuevo; para esto hay que estudiar detalladamente las necesidades de la empresa y las características de los diferentes paquetes en el mercado para ver si se adaptan a éstas y las cubren.

3. EL PROCESO Y EL COSTE DEL MANTENIMIENTO

Flujo del proceso de mantenimiento del software



La estructura de un equipo de mantenimiento acostumbra a ser la siguiente:

- Responsable o líder del equipo: Es el máximo responsable del buen funcionamiento de las tareas de mantenimiento y el interlocutor entre el equipo y el usuario que ha originado la demanda. Tiene conocimientos de todos los sistemas de la empresa y experiencia en liderazgo de grupos de trabajo y en análisis y diseño de programas informáticos.
- Grupo encargado del mantenimiento: Equipo de programadores que se encargan de la modificación de los programas, tiene que haber perfiles especializados similares a los del grupo de desarrolladores.

El número de personas dedicadas al mantenimiento del software no es fijo: la etapa inmediatamente posterior a la entrega del software y las demandas de versiones nuevas marcan los máximos en la actividad de mantenimiento, que va menguando a medida que se da solución a estas peticiones.

El flujo del proceso es el siguiente:

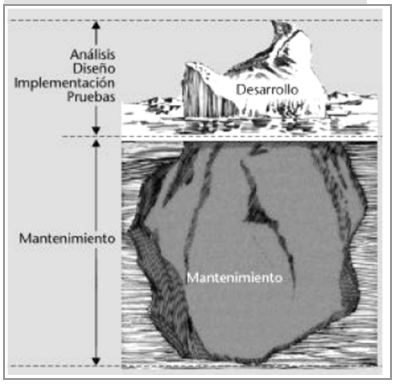
- Llega una petición de corrección de errores del sistema, de modificaciones o de nuevos comportamientos. De esta tarea se suele encargar personal no necesariamente técnico dado que se limitan a recoger incidencias por teléfono o correo electrónico.
- Estas personas remiten las peticiones hacia una persona que supervisa y aprueba o deniega las peticiones de mantenimiento.
- En caso de aceptarse, y de ser una incidencia crítica se asigna un equipo de trabajo para resolver el problema. Si la incidencia no es crítica, se evalúa y clasifica la petición para planificarla correctamente con el resto de peticiones. Se le asigna una prioridad y se pone en cola de peticiones pendientes.
- La persona encargada revisa la tarea de petición de mantenimiento y determina su coste y fecha de inicio.
- El equipo de trabajo estudia la documentación del sistema, la modificación del diseño y la revisión y modificaciones oportunas en el código.

El mismo proceso de mantenimiento, al intentar resolver ciertos problemas, puede generar nuevos errores, los denominados **efectos colaterales**, que pueden afectar al código, a los datos e incluso a la documentación.

Un mantenimiento defectuoso genera una espiral en la que el mantenimiento es cada vez más costoso y peligroso, esta espiral puede generar una situación grave que sólo se puede solucionar por un proceso de reingeniería iniciado en alguna parte del sistema para poder asegurar su mantenibilidad y calidad hasta el final de su vida útil.

Respecto al **coste del mantenimiento del software** decir que tradicionalmente se ha representado con la figura de un iceberg. La parte visible corresponde a las actividades que tienen un coste conocido y previsible: análisis, diseño e implementación. La parte sumergida puede llegar a ser del 80%, correspondiendo a problemas ocultos que tienen asociado un coste alto en esfuerzo y en dinero. Las nuevas tecnologías de la informática han intentado acortar las primeras etapas del ciclo de vida con la creación de lenguajes y herramientas de ayuda de alto nivel. Sin embargo, la mayor parte de las mejoras han ido a parar a las etapas de análisis, diseño e implementación, y el mantenimiento es la parte más abandonada.

Coste de mantenimiento del software



Muchas tareas de mantenimiento se llevan a cabo porque no se han sabido entender las necesidades de los usuarios o porque los usuarios no han sabido definir qué quieren exactamente. En la etapa del mantenimiento se hacen muchas peticiones de mejora o adaptación. El coste de llevar a cabo una modificación varía segú la etapa del ciclo de vida en la que se encuentre el sistema informático: es mucho más caro efectuar un cambio en la etapa de mantenimiento que en las etapas anteriores del ciclo de vida: cuanto más tarde se pide un cambio o se detecta una deficiencia en el sistema, más coste asociado comporta.

Existen **métricas de mantenimiento del software** que pretenden medir este coste, tenemos dos grandes fórmulas:

$$M = p + K^{(c-f)}$$

M= Esfuerzo de mantenimiento P= Esfuerzo productivo
K=Constante empírica f=medida del grado de conoc. Del software
F=Medida de complejidad de mantenimiento atribuible a un diseño erróneo

De esta forma se puede desprender que el esfuerzo total de mantenimiento y su coste, puede crecer exponencialmente si no se utiliza un buen diseño o no se dispone de una buena documentación, o bien las personas encargadas del mantenimiento desconocen el sistema.

La otra métrica se explica de la siguiente forma:

$$IMS = [Mt – (Fa + Fm + Fe)] / Mt$$

Mt= Número de módulos de la versión actual
Fa= Número de módulos de la versión actual añadidos
Fm= Número de módulos de la versión actual modificados
Fe= Número de módulos borrados de la versión anterior

4. EJEMPLOS DE PROCESOS DE MANTENIMIENTO

- **Aparición y modificación del IVA**: La llegada del IVA y su posterior modificación comportó adaptaciones en los sistemas de contabilidad de las empresas. Las empresas que había hecho un buen diseño del sistema, sólo tuvieron que variar el valor de la constante y recompilar todos los programas. Aún tuvieron menos complicación las adaptaciones para las empresas que habían registrado el peso del impuesto en una base de datos: sólo tuvieron que actualizar este dato. Las empresas en cambio, que no habían hecho un buen diseño y habían utilizado el valor directamente en los cálculos internos del código. Tuvieron que repasar todos los programas y, o bien modificar el peso antiguo de la tasa por el nuevo (lo cual resuelve el problema, pero genera otro para el futuro) o bien introducir una constante que sólo se define una vez; este trabajo fue costoso y pesado para los programadores encargados de hacerlo.
- **Efecto 2000**: Los orígenes del efecto 2000 hay que buscarlos en los años 60, cuando dada la capacidad limitada de los ordenadores, era necesario ahorrar todo el espacio posible; una de las maneras de hacerlo fue obviar las dos primeras cifras de los años; no se pensaba que los sistemas tuviesen vidas tan largas. Se tuvieron que localizar todos los programas implicados, y cada programa analizado para encontrar las líneas de código que hacían operaciones con fechas que hubiera que modificar. Este problema no se puede imputar a los programadores que lo originaron: hay que entender las limitaciones tecnológicas con las que se encontraban.
- **Euro**: La adaptación de los programas de contabilidad al euro no ha comportado sólo la variación de un valor por otro o la modificación de un cálculo, se trata de la modificación de un tipo de campo, dado que habitualmente las pesetas no se incluían con decimales y se trataban de campos enteros; además había que posteriormente actualizar todos los valores a la nueva moneda.
- **Absorción de una empresa por otra o fusión**: Tras una absorción o fusión se quiere tener un sistema informático unificado con los datos de las dos empresas. El primer problema que se puede plantear es que los entornos sean distintos: distintas plataformas, sistemas operativos, bases de datos y lenguajes de programación y de explotación de los datos. Las tareas de mantenimiento consistirán en la conversión e integración de los datos en el sistema informático y para ello es necesario conocer a la perfección los dos modelos de bases de datos.

5. EVOLUCIÓN DEL MANTENIMIENTO Y LA REINGENIERÍA DEL SOFTWARE

Los antiguos departamentos informáticos destinan más esfuerzos al mantenimiento de los sistemas que los departamentos más recientes. Esto se explica porque, en general, estos antiguos departamentos dan servicio a sistemas informáticos más grandes y complejos.

Los nuevos métodos y herramientas de diseño, análisis e implementación proporcionan más calidad del software y no requieren tantos esfuerzos de mantenimiento.

El avance generacional de los lenguajes de programación aporta ventajas al desarrollo de nuevo software que facilitan su mantenimiento y aumentan, en general, su calidad.

El importe destinado a mantenimiento ha ido incrementándose en los últimos años, aunque no se debe a un aumento del número de correcciones de errores que, al contrario, han ido disminuyendo, sino por un aumento del número de adaptaciones, cambios y mejoras (debido a que los nuevos sistemas y herramientas de desarrollo son de mayor calidad y que el usuario sabe cada vez mejor qué quiere y cómo lo quiere).

La evolución del software tiene unos objetivos muy claros con respecto al mantenimiento:

- Reducir el mantenimiento correctivo: corregir errores que se podrían haber evitado es un proceso demasiado costoso.
- Maximizar la productividad del mantenimiento adaptativo es la parte que aporta más beneficios a los usuarios.
- Llevar a cabo actividades de mantenimiento perfectivo y preventivo para minimizar los costes de mantenimiento.

La **reingeniería del software** tiene como objetivo la revisión de sistemas y aplicaciones para reacondicionarlos o reconstruirlos para optimizar su operatividad. En la medida en que por medio de la reingeniería se consiga una infraestructura y unos programas más adaptados a las necesidades de los usuarios y más eficientes en coste, tiempos y calidad, las tares de mantenimiento continuas se podrán sustituir por otras de mayor impacto sobre la organización.

Hay que tener en cuenta que una aplicación antigua fue desarrollada teniendo en cuenta las características de la plataforma en la que se debía explotar en su momento, con sus limitaciones. Después de muchas aplicaciones se ha cambiado la plataforma, lo cual ha requerido adaptaciones a los nuevos recursos de hardware y sistemas operativos, pero no se ha modificado su diseño de base. Estas circunstancias con llevar aplicaciones inestables, con un diseño y una lógica inadecuados a la plataforma sobre la que se explotan y un mantenimiento difícil, antieconómico y, a menudo, imposible.

Es importante resaltar la diferencia entre reingeniería y una nueva implementación:

- La reingeniería del software parte de la situación actual de los sistemas de información y se plantea su optimización sin un input externo de nuevas especificaciones.
- La implantación de un nuevo sistema de información o aplicación a la empresa parte de unos requisitos formalizados por parte de los futuros usuarios, o en general, de la organización.

Un proceso de reingeniería del software puede incluir todas o alguna de las actividades siguientes:

- **Análisis del inventario de aplicaciones**: Se debe partir de la identificación y análisis de la situación presente. Todo departamento de tecnología de la información debe disponer, inexcusablemente, de un inventario actualizado de las aplicaciones.
- **Reestructuración de documentos**: Las aplicaciones de cierta antigüedad suelen tener una documentación muy limitada, obsoleta o prácticamente inexistente y, por lo tanto, ineficaz. Dependiendo de la situación de estas aplicaciones podemos pensar en no revisar la documentación y continuar como hasta ahora; en documentar los cambios y las partes de la aplicación a las cuales afectarán los cambios; o documentar toda la aplicación.
- **Ingeniería inversa**: Consiste en analizar una aplicación para obtener su diseño a partir de su código fuente, obteniéndose diagramas con las estructuras de los programas, estructuras de datos, indicaciones de trazabilidad (relaciones entre programas y datos).
- **Reestructuración de datos**: Con frecuencia, más que el código fuente de los programas, lo que condiciona la vida útil de una aplicación y la posibilidad de mantenerla y adaptarla, es la estructura de datos. La reestructuración de datos es un proceso de reingeniería profunda que suele empezar con una ingeniería inversa para obtener los requisitos de la estructura de datos, es uno de los procesos más delicados de la reingeniería del software y comporta un riesgo muy elevado: afecta a los datos, a los programas y al conjunto de los sistemas de información, dado el elevado grado de integración entre aplicaciones. Hay

dos razones fundamentales que llevan a plantearse la necesidad de una reestructuración de datos:

  - La tendencia actual hacia un incremento del protagonismo de los usuarios en el acceso a la información y la preparación de informes, que comporta una necesidad creciente de disponer de bases de datos ordenadas y accesibles de manera lógica.
  - El dimensionamiento: a veces los programas que se desarrollan para manejar un número de casos limitado, quedan desbordados por la expansión en los elementos que debe gestionar y, en definitiva, por el volumen de datos que tienen que manipular.

- Cuantificación de costes y beneficios: Hemos comprobado que la reingeniería del software requiere casi siempre un volumen elevado de recursos y representa un coste importante. Por ello es imprescindible analizar la relación costes/necesidad/beneficio antes de emprender un proceso de reingeniería del software.

Texto elaborado a partir de:
Proceso de ingeniería del software
Maria Jesús Marco Galindo, Eulàlia Clos Cañellas, Isabel Doménech Puig-Serra, Àlex Alfonso Minguillón, Humberto Andrés Sanz, Jordi Schoenenberger Arnaiz
Febrero 2007