Proceso de software
Es un conjunto de actividades y resultados asociados que generan en un producto de software.
Existen diferentes modelos de proceso, que son formas de representar de manera abstracta un proceso (y por extensión las actividades que lo componen).
Actividades asociadas al proceso de desarrollo (sea por modelos tradicionales o metodologías ágiles):

- Especificación del software
- Elicitación de requerimientos
- Especificación de requerimientos
- Desarrollo del software
- Validación de software
- Evolución del software

Comunicación
Peter Heinemann dice que la comunicación es un proceso de interacción social a través de símbolos y sistemas de mensajes que se producen como parte de la actividad humana.
Es el proceso más importante de la interacción humana (personalmente agregaría que no hay interacción humana posible sin alguna forma de comunicación, al menos según la interpretación de Peirce de los signos, según la cual toda nuestra manipulación conceptual es a través de signos codificados), una necesidad personal y la forma más completa de crear la realidad. Las personas comunicamos constantemente, tanto con las palabras (comunicación verbal) como con nuestra actitud y comportamiento (comunicación no verbal - gestos, postura, etc.)
Elicitación
Es el proceso de adquirir (“eliciting”, en inglés sonsacar) todo el conocimiento relevante necesario para producir un modelo de los requerimientos de un dominio problema
Objetivos:

- Conocer el dominio del problema para poder comunicarse con clientes y usuarios y entender sus
- necesidades
- Conocer el sistema actual (manual o informatizado)
- Identificar las necesidades, tanto explícitas como implícitas, de clientes y usuarios y sus expectativas sobre el sistema a desarrollar.

Técnicas:
Recolección de hechos a partir de la documentación existente
Documentos que pueden enseñar algo acerca del sistema:

- Organigrama: identifica al propietario, usuarios claves, y la estructura del sistema como el cliente la concibe
    - Memos, notas internas, minutas, registros contables: dan acceso a una visión más real del
    - sistema (en oposición a la conceptual del organigrama)
    - Solicitudes y documentación de proyectos de sistemas de información anteriores: permiten
    - conocer el historial que origina el proyecto.
- Observación del ambiente de trabajo: el analista se convierte en observador de las personas y actividades con el objetivo de aprender acerca del sistema
  Lineamientos de la observación:
    - Determinar quién y cuándo será observado
    - Obtener el permiso de la persona y explicar por qué será observado
    - Mantener bajo perfil (observación no participante)
    - Tomar nota de lo observado
    - Revisar las notas con la persona apropiada
    - No interrumpir a la persona en su trabajo
- Investigación del dominio: puede incluir una o varias de las siguientes acciones
    - Visitas al sitio
    - Recopilación de patrones de soluciones (mismo problema en otra organización)
    - Lectura de revistas especializadas
    - Búsqueda de problemas similares
    - Consulta a otras organizaciones
- Cuestionarios
- Entrevistas: pasos

- Lectura de antecedentes: poner atención en el lenguaje. Buscar un vocabulario en común. Imprescindible para poder entender al entrevistado.
- Establecer objetivos de la entrevista
- Selección de entrevistados
- Planificación de la entrevista

Comunicación de ideas

Elevator pitch

Elevator pitch o venta de ascensor, es la idea de comunicar una idea en poco tiempo (unos pocos pisos dentro de un ascensor, por ejemplo) a un potencial cliente/inversor, y convencerle de que participe de la misma (con una inversión económica, prestando servicios o tecnología de manera gratuita o a un costo reducido, etc.). Al comunicar una idea o proyecto, lo importante es ser conciso y buscar el mayor impacto posible en poco tiempo:

- Iniciar con un disparador relevante y de alto impacto, que permita entender la necesidad que da origen al proyecto (cuál es el problema que nuestro proyecto soluciona)
- Continuar con la idea, comunicándola como si ya estuviera resuelta (debería estar lo más cerrada posible)
- Concluir con algún tipo de call to action: concretar una reunión, entrevista, o invitación a participar.

Los pasos de un elevator pitch

1. Afirmación o pregunta sorprendente para llamar la atención
2. Presentación: contar quién sos
3. Problemas o necesidades que cubrís
4. ¿Qué soluciones aportás?
5. ¿Cuál es el principal beneficio que obtiene la gente con vos?
6. ¿Por que tu proyecto es el idóneo? (Cómo sos mejor que la competencia, por qué al inversor le conviene poner plata en tu idea antes que invertirla de otra forma)
7. Call to action

Tips de comunicación:

No verbal:

- Las personas deben sonreir y/o aparecer relajadas y confiadas
- Utilizar los dedos para contar ideas o puntos importantes
- Mover las manos para dar énfasis o apoyar el relato verbal

Verbal:

- Ajustar el tono y la modulación de la voz para comunicar efectivamente y resaltar unas cosas sobre otras
- Analizar cuándo modificar la voz y cuándo utilizar gestos no verbales

Cómo construir un proyecto o idea

- ¿Qué es lo primero que debe hacerse para arrancar la presentación de la idea/proyecto?
- ¿Cómo se ofrece el proyecto?
- ¿Cómo sugiere el video que se consiga el financiamiento?
- ¿Cómo se sugiere continuar la presentación del proyecto?

Powerpoint

Al momento de realizar una presentación con un software tipo powerpoint:

menos es más & show don’t tell

Reglas de presentación (Guy Kawasaki):

10/20/30

- Máximo 10 diapositivas
- Máximo 20 minutos de duración
- Mínimo cuerpo 30 en los textosAl momento de presentar:
- No leer la presentación al pie de la letra: la presentación es un apoyo visual a lo que yo tengo para decir, yo debería tener un discurso, algo importante y relevante que deseo comunicar, con lo que estoy familiarizado y comprometido. La presentación tiene el fin de funcionar como contrapunto y énfasis de mi discurso.
- No mirar la pantalla al exponer: relacionado con lo anterior, la mirada del presentador debe estar enfocada en su público, no en las notas de la presentación. Es importante saber qué decir en cada momento y que el público pueda leer en nosotros confianza y familiaridad con nuestro mensaje

Consideraciones de diseño

- Una sóla idea por diapositiva
- No animar el texto de las diapositivas
- Usar un tipo de letra legible
- Fondos claros y sencillos
- Titulares, no apartados
- Cuidado con los bullets

Se recomienda cerrar la presentación con una sección de preguntas para despejar las dudas que la presentación pueda generar en el público (es importante estar preparado para esta etapa, la presentación puede construirse también con algunos huecos evidentes pero no cruciales (o parecerá poco profesional), que ayuden a disparar dudas y generen engagement).
Después de la sesión de dudas, cerrar con un resumen de lo expuesto
Tips adicionales:

- La presentación debe ser llamativa desde lo visual, debe ofrecer un contenido disruptivo o novedoso
- Debe invitar a ser vista por quien recibirá el mensaje
- El contenido debe ser afirmativo en tiempo presente, por ejemplo: “esta app permite...”
- No deben faltar:
  - Nombre del proyecto
  - Nombre de la empresa de desarrollo
  - Objetivo
  - Características o funcionalidades principales y/o diferenciadoras
  - Técnicas de recolección de datos que se emplean
  - Información de contacto

Requerimientos / SRS
Etapas de requerimientos:

- Solicitud/elicitación (anglicismo de eliciting)
- Definición
- Análisis
- Especificación

Especificación
Documento dirigido a los usuarios
A través de un documento llamado ConOps, normalizado en el estándar IEEE Std. 1362-1998. Describe las características de un sistema propuesto desde el punto de vista de los usuarios.
La Descripción del Sistema, es el medio de comunicación que recoge la visión general, cualitativa y cuantitativa de las características del sistema; compartido por la parte cliente y desarrolladora.
El IEEE ofrece un formato y contenidos para la confección de las descripciones de sistema en los desarrollos y modificaciones de sistemas. El estándar no especifica técnicas exactas, si no que proporciona las líneas generales que deben respetarse - es una guía de referencia. A su vez, identifica los elementos mínimos que debe incluir una Descripción del sistema, pudiendo el usuario agregar cláusulas y sub-cláusulas.
Documento dirigido al desarrollo
Comúnmente denominado SRS (sigla de Software Requirements Specification, o en español ERS por Especificación de Requerimientos de Software) normalizado en el estándar IEEE Sts. 830-1998.
Es la especificación de las funciones que realiza un determinado producto de software, programa o conjunto de programas en un determinado entorno.
El documento de especificación de requisitos puede desarrollarlo personal representativo de la parte desarrolladora, o de la parte del cliente; aunque es aconsejable la intervención de ambas partes.
Alcance del estandar: brindar una colección de buenas prácticas para escribir especificaciones de requerimientos de software. Se describen los contenidos y cualidades de una buena especificación de requerimientos
Naturaleza del SRS: es una especificación para un producto de software particular
Ambiente del SRS: el software puede contener toda la funcionalidad del proyecto, o puede ser parte de un sistema más grande. En el último caso habrá un SRS que declarará las interfaces entre el sistema y su software desarrollado, y pondrá qué función externa y requerimientos de funcionalidad tiene con el software desarrollado.
Este documento debe cumplir con las siguientes características:

- Correcto: cada requisito declarado se encuentra en el software
- No ambiguo: cada requisito tiene una única interpretación
- Completo: reconoce todos los requisitos externos impuestos por especificaciones del sistema
- Consistente: se refiere a la consistencia interna (dentro del proyecto) debe estar de acuerdo con toda la documentación pertinente al proyecto (tanto con documentación de mayor jerarquía, como una normativa, como de menor jerarquía, como una especificación de requerimientos en concreto)
- Priorizado: evalúa la prioridad de los requerimientos particulares
- Comprobable: todos los requerimientos deben ser comprobables, condición que sucede si y sólo si existe algún proceso con el que una persona o máquina puede verificar que el producto del software reúne el requisito. En general, cualquier requisito ambiguo no es comprobable.
- Modificable: el documento debe ser susceptible de recibir modificaciones en los requerimientos con facilidad, sin que esto implique pérdidas de consistencia en información, estructura o estilo.
- Trazable: debe indicarse claramente el origen de cada requerimiento, y estos deben ser susceptibles de trazarse hacia otros requerimientos de futuros desarrollos.

Consideraciones para un buen SRS

- Preparación conjunta: todas las partes intervinientes deben participar en la confección del SRS, a fin de garantizar un buen acuerdo entre los participantes.
- Evolución: el SRS debe evolucionar conjuntamente con el software, registrando los cambios, los responsables y aceptación de los mismos.
- Prototipos: los prototipos son una herramienta extremadamente útil para la definición de requerimientos
- Diseño: el SRS puede incorporar atributos o funciones axternos al sistema, en particular aquellas que describen el diseño para interactuar entre los subsistemas.
- Requerimientos: los detalles particulares de los requerimientos son anexados como documentos externos (CU, Plan de proyecto, Plan de aseguramiento de la calidad, etc.)

Estructura del documento

1. Introducción

1.1. Propósito: define el propósito del documento y especifica a quién va dirigido el documento.
1.2. Alcance o ámbito del sistema: se da un nombre al futuro sistema. Se explica lo que el sistema hará y lo que no hará. Se describen los beneficios, objetivos y metas que se espera alcanzar con el futuro sistema.
1.3. Referencias: se presenta una lista completa de todas las referencias de los documentos mencionados o utilizados para escribir el SRS. Se identifica cada documento por el título, número de reporte, fecha y publicación, así como las fuentes de las referencias de donde se obtuvieron.

2. Descripción general

Esta sección debe describir los factores generales que afectan el producto y sus requerimientos. No declara los requerimientos específicos, ya que estos se definen en detalle en la sección 3 del SRS. Normalmente aquí se abarcan

2.1. Perspectiva del producto

Si el producto es independiente y autónomo debe declararse que así es. Por otro lado, si el SRS define un producto que es un componente de un sistema más grande, se debe relacionar los requerimientos de este ultimo con la funcionalidad del software, y se deben identificar las interfaces involucradas.

2.2. Funcionalidades del producto

Se debe presentar un resumen de las funciones futuras del sistema. Deben mostrarse de forma organizada, y puede hacerse a través de gráficos, siempre que reflejen las relaciones entre funciones, y no el diseño del sistema (CU)

2.3. Características de los usuarios

Deben describirse las características generales de los usuarios a quienes está dirigido el producto, esto incluye: nivel educativo, experiencia y especialización técnica.

2.4. Evoluciones previsibles del sistema

Se identifican requerimientos que serán implementados en futuras versiones del software

3. Requerimientos no funcionales

Debe contener todos los requerimientos no funcionales del software a un nivel de detalle que permita a los diseñadores diseñar el sistema, y a los auditores poder comprobar si el sistema los satisface.
3.1. De rendimiento
Relacionados con la carga de uso que se espera tenga que soportar el sistema, como cantidad de terminales, o de usuarios conectados en simultáneo. Todos los requerimientos deben ser mensurables; por ejemplo: “el 95% de las transacciones deben realizarse en menos de 1 segundo”.
3.2. Seguridad
Especifica los elementos que protegerán al software de accesos, usos y sabotajes maliciosos, así como de modificaciones o destrucciones maliciosas o accidentales. Se puede especificar: empleo de técnicas criptográficas, registro de ficheros con “logs” de actividad, asignación de determinadas funcionalidades a determinados módulos, restricción de comunicación entre determinados módulos, comprobaciones de integridad de información crítica, entre otros.
3.3. Portabilidad
Atributos que debe presentar el software para facilitar su traslado a otras plataformas o entornos. Puede incluirse: % de componentes dependientes del servidor, % de código dependiente del servidor, uso de un determinado lenguaje por su portabilidad, uso de un determinado compilador o plataforma de desarrollo, uso de un determinado sistema operativo.
4. Mantenimiento
- Identificación del tipo de mantenimiento necesario del sistema
- Especificación de quién debe realizar las tareas de mantenimiento, por ejemplo usuarios, o un desarrollador.
- Especificación de cuándo deben realizarse las tareas de mantenimiento, por ejemplo: generación de estadísticas de acceso semanales y mensuales.
5. Apéndices
Pueden contener todo tipo de información relevante para la SRS, pero que no forman parte de su estructura general, como pueden ser casos de uso e historias de usuario.

Gestión de riesgos

Riesgos

Definición: un riesgo es un evento no deseado, y cuya presencia no puede identificarse con certeza, que tiene consecuencias negativas.
Todo proyecto de desarrollo conlleva riesgos, más o menos predecibles, y con mayor o menor grado de impacto sobre la viabilidad y vida del propio proyecto. Por lo tanto, los gerentes deben determinar qué eventos pueden presentarse durante el desarrollo o el mantenimiento, y hacer planes para evitarlos o minimizar sus consecuencias negativas. En este sentido, los aspectos importantes del proceso son la anticipación y planificación.
Características inherentes a la definición de riesgo, son:

- Incertidumbre: el grado de probabilidad de que el riesgo se manifieste (nunca puede ser 100%, si no se trata de un problema previsible y que debe ser considerado durante la planificación)
- Pérdida: si el riesgo sucede, tendrá un impacto negativo estimable sobre el proyecto

Los riesgos pueden abarcar:

- Cómo pueden afectar cambios en el proyecto al desarrollo del mismo (cambios en requerimientos, por ejemplo).
- Las elecciones tomadas respecto a la planificación del desarrollo y mantenimiento (tamaño y especialización de equipos - personal -; herramientas, tecnologías y lenguajes a utilizar, etc.).
- Sucesos relativamente difíciles de determinar que puedan acontecer en el futuro (una pandemia global sería un caso extremo, pero evidentemente no ridículo de considerar).

Clasificaciones de riesgo

Según la perspectiva de análisis, los riesgos se pueden agrupar de diferentes formas. Una de ellas está relacionada al aspecto del proyecto sobre el que impacta o que da origen al riesgo:

- Riesgo relacionado al proyecto: problemas de planificación y gestión
- Riesgo relacionado al producto: problemas técnicos o tecnológicos
- Riesgo relacionado al negocio: problemas inherentes a la relación comercial que lleva al desarrollo del proyecto, o modificaciones en el ámbito de aplicación del mismo

Otra forma de clasificar a los riesgos es según su tipo en dos categorías:

- Genéricos: son susceptibles de suceder en cualquier proyecto. El cambio de requerimientos, por ejemplo, es un riesgo de este tipo.
- Específicos: su aparición dependerá del dominio y características específicas del proyecto. Modificaciones en legislación sobre el dominio en el que se aplica el proyecto podrían ser ejemplo de este tipo (aunque afectan a los requerimientos del software, lo hacen a por características específicas del proyecto)

Esta división en tipos, a su vez, contempla para ambas categorías una subdivisión basada en su predecibilidad, es decir, no en la probabilidad de que sucedan, si no en qué tan fácil es reconocerlos. La subdivisión es en tres categorías:

- Conocidos: quienes trabajan en proyectos de este tipo (en el sentido amplio de desarrollo de software en casos genéricos, y en lo concreto de un dominio en casos específicos) están familiarizados con la aparición y gestión de estos riesgos.
- Predecibles: estos riesgos no son necesariamente tan comunes, pero son susceptibles de preverse.
- Impredecibles: estos riesgos muy difícilmente puedan ser identificados.

Estrategias

Reactivas

Implican un reconocimiento de la posibilidad de aparición de riesgos, pero la manera de encararlos consiste en trabajar alrededor de ellos una vez que se manifiestan. En otras palabras, se trata de “gestionar la crisis”.

Proactivas

Implican el desarrollo de planes de acción predefinidos para los riesgos identificados, y su puesta en marcha si es que uno de dichos riesgos se manifiesta.

Proceso de Gestión de Riesgos

En caso de asumir una estrategia proactiva, debe llevarse a cabo un proceso de gestión de riesgos. Éste puede llevarse a cabo según la siguiente estructura:

1. Identificación: el equipo genera un listado de riesgos potenciales
2. Análisis: se estudian los riesgos listados en la etapa anterior, asignándoles un grado de impacto y un nivel de probabilidad. A partir de allí se genera un criterio de priorización de riesgos, según el cual se ordena esta lista. Finalmente, se decide una línea de corte, que definirá el conjunto de riesgos que se consideran “relevantes”.
3. Planeación: se consideran y describen las estrategias necesarias para afrontar los riesgos definidos.
4. Supervisión de riesgos: se efectúa un control sobre el proceso, evaluando si los riesgos suceden, si aparecen nuevos, y si las estrategias seleccionadas resultan útiles y en qué grado. La supervisión implica una re-valoración de riesgos y un retorno cíclico a las etapas anteriores, por lo que el proceso de gestión de riesgos es en realidad iterativo.

Identificación

Se genera una lista de aquellos que pueden ser considerados “verdaderos riesgos” es decir aquellos que desde un principio es notorio que pueden afectar al proyecto en un grado considerable.

Un buen punto de partida puede ser la utilización de una lista de comprobación de elementos de riesgo, que genera seis categorías disparadoras para la inspección y definición de riesgos:

1. Tecnológicos
2. Personales
3. Organizacionales
4. De herramientas
5. De requerimientos
6. De estimación

A su vez, se puede recurrir a una serie de preguntas que indagan sobre posibles aristas débiles del proyecto, o puntos en los que puede haber riesgos no identificados. Un ejemplo de ello sería la siguiente lista:

¿Los ingenieros de software y el cliente se reunieron formalmente para apoyar el proyecto?
¿Los usuarios finales se comprometen con el proyecto y sistema/producto que se va a construir?
¿El equipo y sus clientes entienden por completo los requisitos?
¿Los clientes se involucraron plenamente en la definición de los requisitos?
¿Los usuarios finales tienen expectativas realistas?
¿El ámbito del proyecto es estable?
¿El equipo tiene la mezcla correcta de habilidades?
¿Los requisitos del proyecto son estables?

¿El equipo tiene experiencia con la tecnología que se va a implementar?
¿El número de personas que hay en el equipo es adecuado para hacer el trabajo?
¿Todos los clientes/usuarios están de acuerdo en la importancia del proyecto y en los requisitos para el sistema/producto que se va a construir?
Si la respuesta a cualquiera de estas preguntas es negativa, estamos frente a uno o varios riesgos inminentes, en cuyo caso el grado de riesgo total es directamente proporcional a la cantidad de respuestas negativas.
Esta tarea puede ser llevada a cabo mediante una metodología de tormenta de ideas, pero será enriquecida principalmente por la experiencia de los participantes.
Cada riesgo identificado debe ser clasificado según las categorías de origen mencionadas anteriormente: proyecto, desarrollo y negocio, y a su vez según su nivel de predictibilidad (conocido, predecible, impredecible).

Análisis

Una vez identificados y categorizados los riesgos, se definen la probabilidad e impacto de cada uno de ellos. Con esto hecho se confecciona la tabla de riesgos, con el siguiente formato:

Riesgo	Categoría/s	Probabilidad estimada (1)	Impacto
Una lista de cada riesgo sin un orden específico	La categoría o categorías a las que corresponde cada riesgo	La estimación es principalmente heurística/intuitiva. Puede establecerse por consenso, o realizarse una estimación individual por parte de cada miembro del equipo, y luego un promedio de las aproximaciones.	Similar al caso anterior, aunque puede describirse también los puntos sobre los que impactará principalmente el riesgo.

Dado que la probabilidad de un riesgo nunca puede ser determinada con precisión, puede ser conveniente establecer una de aproximaciones, a fin a su vez de poder agrupar más fácilmente los riesgos según este parámetro.
Una escala posible es:
Bastante improbable: < 10%
Improbable: 10-25%
Moderado: 25-50%
Probable: 50-75%
Bastante probable: >75%
Esta noción también puede ser aplicada a la evaluación de impacto. La siguiente es una posible escala para este fin:

1. Catastrófico: cancelación del proyecto
2. Serio: reducción de rendimiento, retrasos en la entrega, exceso importante en costo
3. Tolerable: reducciones mínimas de rendimiento, posibles retrasos, exceso en costo
4. Insignificante: incidencia mínima en el desarrollo

Una vez completada la tabla, incluyendo categorías, probabilidades e impacto, ésta debe ordenarse en función de su probabilidad e impacto. Finalmente, se traza una línea de corte.

Criterios de orden

Si bien hasta aquí se establecieron una probabilidad y un impacto para cada riesgo, las posibles combinaciones de estos valores no necesariamente permiten ordenar los riesgos según su prioridad de manera evidente. Aún así, pueden establecerse ciertos criterios elementales para desarrollar este ordenamiento:

- Si un factor de riesgo tiene gran impacto, pero poca probabilidad de ocurrir, no debería absorber un tiempo significativo.
- Es recomendable tomar en cuenta los riesgos con mayor impacto y una probabilidad moderada o superior, y los riesgos con poco impacto pero con gran probabilidad.

Línea de corte

Hacer un seguimiento y planificación de todos los riesgos identificados puede convertirse en una tarea muy compleja, y sobre algunos riesgos (aquellos con menor probabilidad e impacto), es probable que no sea

necesario hacer demasiado. Por ello es recomendable hacer una selección de los riesgos de mayor relevancia, y limitar el seguimiento a estos.
La pregunta que surge entonces es ¿A cuántos riesgos se les debería seguimiento? O, en otras palabras ¿Dónde se coloca la línea de corte? En este sentido, toda selección resultará arbitraria (Bohem, por ejemplo, recomienda identificar y supervisar los 10 riesgos más importantes), pero existirá un mínimo de información contextual dada por la naturaleza del proyecto, así como la intuición y experiencia de los ingenieros, que permitirá definir aproximadamente a cuáles y cuántos riesgos realmente resulta conveniente hacer un seguimiento a conciencia. Uno de los aspectos coyunturales a tener en cuenta es la cantidad y complejidad de riesgos a los que el equipo tiene la capacidad efectiva de hacer un seguimiento activo.
Una vez establecida la línea de corte, aquellos riesgos que queden por encima serán los que reciban la mayor atención, mientras que los demás serán reevaluados y tendrán una prioridad de segundo orden.

Administración

Las estrategias seleccionadas para esto pueden ser de tres tipos:

- Anulación: se toman las medidas necesarias para que el riesgo no pueda impactar sobre el proyecto. Ej: si hay riesgos de problemas de suministro de energía eléctrica se instalan grupos electrógenos y estabilizadores de tensión o UPS.
- Minimización: si un riesgo no puede ser anulado, pero se pueden tomar medidas de forma preventiva, se hace esto último. La minimización en este caso se refiere a minimizar la probabilidad de que sucedan. Ej: se garantiza la existencia de desarrolladores idóneos que puedan suplir eventuales ausencias de los miembros del equipo.
- Planificación de contingencia: si un riesgo no puede ser anulado ni minimizado, se genera un plan de acción para el caso en que el riesgo se manifieste, a fin de minimizar las consecuencias del mismo.

Otro aspecto a considerar al momento de tratar los riesgos, es el costo de aplicación de las estrategias. Por este criterio, en muchas ocasiones se emplean exclusivamente estrategias reactivas.

Supervisión

Los riesgos deben monitorizarse en común en todas las etapas del proyecto. En cada una de estas instancias, es necesario estudiar cada uno de los riegos por separado, considerando una reevaluación de su probabilidad e impacto. A su vez, si un riesgo se manifestó, debe analizarse la efectividad de la o las estrategias empleadas para contrarrestarlo. Debe garantizarse también que todas las medidas necesarias para enfrentar cada riesgo están siendo tomadas.
No menos importante es recopilar toda la información existente sobre riesgos acontecidos o no, a fin de contar con datos concretos que puedan informar la toma de decisiones a futuro.

Gestión de la Configuración del Software (GCS)

En el momento de construcción de software, es común que sucedan cambios en distintos componentes y etapas del desarrollo, sea del software mismo (cambios en los requerimientos) o en alguna otra documentación, tarea o actividad. Independientemente de su aparente relevancia, es importante hacer un buen control de estos cambios, al menos reconociéndolos y a sus ramificaciones, y actuando en consecuencia con esa información. De otra forma, se corre el riesgo de incrementar sustancialmente el nivel de confusión en el equipo de desarrollo y, como consecuencia, en el usuario final. La mala gestión de modificaciones conforma, en sí misma, un riesgo en el proceso de software.
La gestión de la configuración de software es esencialmente el proceso de identificar y definir los elementos del sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando y reportando el estado de los elementos y las solicitudes de cambio, y verificando que los elementos estén completos y sean los correctos. Funciona principalmente como una actividad de autoprotección a ser aplicada durante el proceso de software. Algunos de los elementos de la configuración (es decir aquellos sobre cuyas modificaciones la GCS puede tener ingerencia) son:

- Especificación del sistema
- Plan del proyecto de software
- Especificación de diseño
- Diseño preliminar
- Diseño detallado
- Listados del código fuente
- Planificación y procedimiento de prueba
- Casos de prueba y resultados registrados
- Manuales de operación y de instalación

- Programas ejecutables
- Descripción de la base de datos
- Esquema, modelos
- Datos iniciales
- Manual de usuario
- Documentos de mantenimiento
- Estándares y procedimientos de ingeniería de software

En todos estos aspectos (y otros) pueden aparecer cambios en cualquier momento; las actividades de la GCS sirven para:

- Identificar el cambio (origen, repercusiones, etc.)
- Controlar el cambio (reducción de riesgos, seguimiento)
- Garantizar que el cambio se implemente adecuadamente
- Informar el cambio a todos aquellos que puedan estar afectados

Cada elemento de la configuración del software que se conforma como una línea base, debe ser gestionado formalmente de esta manera por la GCS.

Línea base (baseline): Una línea base es un acuerdo entre todas las partes sobre una o más características del producto final.

Según IEEE:

Una especificación o producto que se ha revisado formalmente y sobre el que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de controles de cambio.

En el contexto de la IS:

Una línea base es un punto de referencia en el desarrollo de software que queda marcado por el envío de uno o más elementos de la configuración de software (ECS) y su aprobación.

Importancia de la GCS

La GCS establece formalmente las respuestas a estos y otros interrogantes que pueden aparecer al momento de estudiar la gestión de cambios:

¿Cómo identifica y gestiona una organización las diferentes versiones existentes de un programa (y su documentación) de forma que se puedan introducir cambios eficientemente?

¿Cómo controla la organización los cambios antes y después que el software sea distribuido al cliente?

¿Quién tiene la responsabilidad de aprobar y asignar prioridades a los cambios?

¿Cómo podemos garantizar que los cambios se han llevado a cabo adecuadamente?

¿Qué mecanismo se usa para avisar a otros de los cambios realizados? ¿Cómo se define a quién se le deben comunicar estos cambios?

Proceso de GCS

El proceso de GCS se divide en cinco partes elementales:

1. Identificación de los objetos en la GCS

Se refiere a la nomenclatura, es decir a otorgar un nombre o identificador a los elementos pertinentes a la GCS. Esta identificación implica:

- Nombre: una cadena de caracteres no ambigua y que caracteriza unívocamente al objeto.
- Descripción: una lista de elementos de datos que identifican:
- Tipo de ECS (documento, código fuente, datos, etc.)
- Identificador del proyecto
- Información de la versión y/o cambio

2. Control de versiones

Es la combinación de procedimientos y herramientas para gestionar las versiones de los ECS que se crean a lo largo del proceso de software. La idea principal es la de contar con un historial de versiones de los diferentes archivos (no sólo archivos de código, si no todos los documentos y otros artefactos que puedan ser generados en el proceso de desarrollo), que permita revisar y regresar a versiones anteriores.

Se entiende por versión de un sistema a una instancia del mismo que difiere en una o más de sus características de otras instancias. Puede incluirse en el control de versiones una gestión de la relevancia de los cambios que existen entre diferentes instancias (de 4.2 a 4.3 sería esperable que haya cambios menos sustanciales que de 4.3 a 5.1)Ejemplo:

Un programa dado contiene los módulos 1, 2, 3, 4, 5

Una versión utiliza los módulos 1, 3, 4, 5

Otra versión utiliza los módulos 2, 3, 4, 5
Con lo que se tendrían dos variantes de un mismo programa, cada una de las cuales debería estar identificada (y gestionados los cambios entre una y otra)

Glosario

La tarea de control de versiones trae asociado un conjunto de vocablos específicos que es importante conocer

- Repositorio (informal repo): el paquete o espacio en el que se almacenan los archivos actualizados e históricos del proyecto.
- Versión: determina un conjunto de archivos que conforman una instancia de ese registro histórico
- Main o master (Principal): el conjunto de archivos principales del proyecto, o núcleo base del mismo.
- Apertura de rama (branch): punto en el que una línea de modificaciones históricas se separa de la línea principal (una rama puede en realidad salir de otra rama). A su vez, rama o branch es el nombre de una línea histórica separada de la principal pero con un punto común en el pasado.
- Commit (compromiso/comprometerse): proceso de confirmación de cambios locales al repositorio (se lo podría considerar equivalente a “guardar los últimos cambios”)
- Conflicto: problema entre las versiones de un mismo documento, por lo general en diferentes copias de una misma versión (locales vs repositorio, o diferentes copias locales).
- Cambio (diff): representa una modificación específica
- Integración (merge): proceso de fusión de dos ramas del proyecto, recibe el mismo nombre el punto de convergencia resultante.
- Actualización (sync o update): integra los cambios que han sido hechos en el repositorio y las copias locales que puedan existir de éste.

3. Control de cambios

Como se mencionó anteriormente, los cambios a lo largo del proyecto son algo inevitable, y su control es una tarea clave para el buen desarrollo del mismo. Esta parte del proceso se ocupa de la combinación de los elementos humanos y las herramientas técnicas y tecnológicas adecuadas para proporcionar un mecanismo formal para el control de cambios.

Entre otras cosas establece la necesidad de contar con una autoridad de control de cambios (ACC) que se ocupa de revisar y aprobar o denegar los cambios propuestos por usuarios, desarrolladores, técnicos u otros agentes vinculados al desarrollo. Dependiendo de la envergadura del proyecto, la ACC puede estar conformada por una o más personas idóneas (especialistas del dominio, ingenieros de software, analistas de sistemas, gerentes de control de calidad, etc.). Algunas de las cuestiones que evalúa la ACC, son:

- ¿Cómo impactará el cambio en el hardware?
- ¿Cómo impactará el cambio en el rendimiento?
- ¿Cómo alterará el cambio la percepción del cliente sobre el producto?
- ¿Cómo afectará el cambio la calidad y la fiabilidad?

4. Auditoría de la configuración

Para garantizar que un cambio se ha realizado correctamente (posteriormente a su identificación, control de versiones y cambios), se deben realizar tanto revisiones técnicas formales, como auditorías de la configuración. Estas se encargan de responder a las siguientes interrogantes:

- ¿Se ha hecho el cambio especificado en la Orden de Cambio? ¿Se han incorporado modificaciones
- adicionales?
- ¿Se ha llevado a cabo una revisión técnica formal (RTF) para evaluar la corrección técnica?
- ¿Se han seguido adecuadamente los estándares de IS?
- ¿Se han reflejado los cambios en el ECS (fecha, autor, atributos)?
- ¿Se han seguido procedimientos de GCS para señalar el cambio, registrarlo y divulgarlo?
- ¿Se han actualizado adecuadamente todos los ECS relacionados?

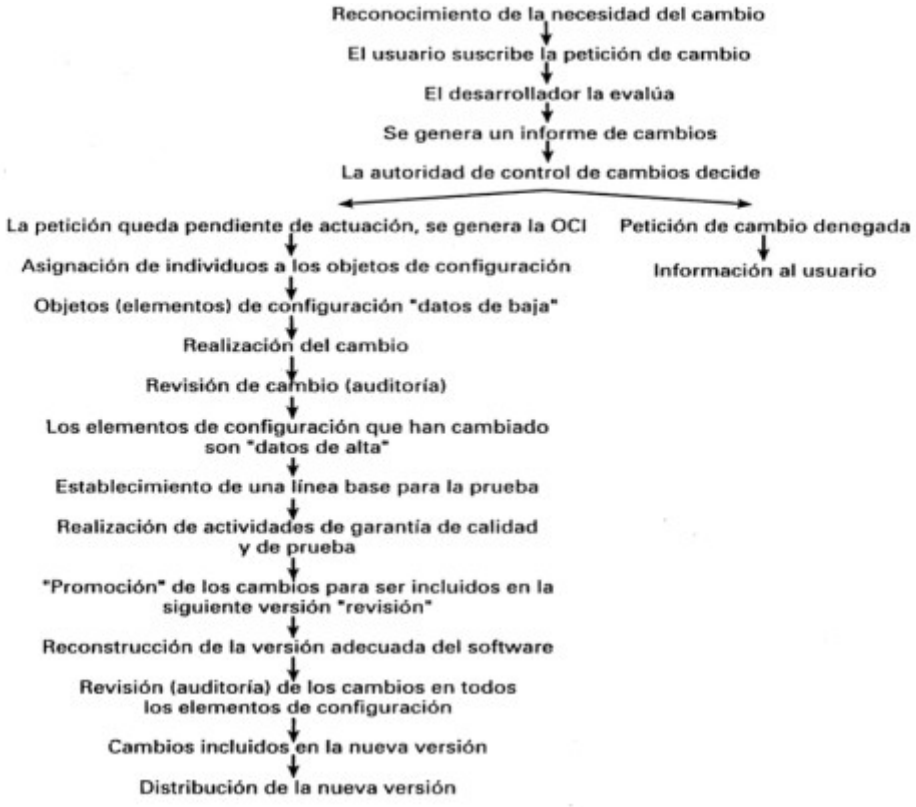
5. Generación de informes de estado de la configuración

De la misma forma que se debe contar con un registro histórico de los ECS, es importante también contar con un registro de similares características sobre la propia GCS. Para ello se generan periódicamente (posteriormente a algún cambio significativo) informes de estado de la configuración. Éstos informes indican:

- Qué pasó
- Quién lo llevó a cabo
- Cuándo sucedió

- Qué se vio afectado directa e indirectamente

El proceso de un cambio sigue más o menos la siguiente estructura:



Gestión de proyectos

Un proyecto es un esfuerzo desarrollado a lo largo de un tiempo determinado no instantáneo, que se lleva a cabo para crear un producto, servicio o resultado único. Se caracteriza por:

- Tener un comienzo y fin definidos
- Contar con salidas o resultados a su término (productos, servicios, etc.)
- Desarrollarse gradualmente

En el contexto particular de la IS, la gestión de un proyecto consiste en la subdivisión de un proyecto de software en un conjunto ordenado de tareas más sencillas que facilitan el desarrollo y su seguimiento. Esta actividad engloba o está comprendida por cuatro aspectos:

- Planificación
- Dirección
- Organización
- Control

Como generalización, se puede decir que una mala gestión de proyectos tiene algunos indicadores típicos:

- Incumplimiento de plazos
- Incremento de costos
- Baja calidad del producto final

En todas sus aristas, la administración efectiva de un proyecto de software debe enfocarse, en orden decreciente de importancia (pero sin por ello desatender a ninguna), en lo que se llaman las 4P de la Gestión de Proyectos:

- Personal: es el componente que hace el esfuerzo concreto para llevar adelante el desarrollo. El equipo de gestión del proyecto debería identificar a los involucrados, sus habilidades, limitaciones y expectativas, y administrarlos para alcanzar un proyecto exitoso.
- Producto: al ser un intangible, a veces es difícil ver el progreso del proyecto. Para contrarrestar esta situación es importante generar una buena planificación, estableciendo objetivos concretos y comprobables para las diferentes etapas, que permitan hacer un seguimiento precisamente objetivo del avance del proyecto. Debe establecer además las técnicas y herramientas que cada etapa del proyecto requerirá, y cómo deben ser administradas.

- Proceso: proporciona el marco de trabajo desde el cual se puede establecer un plan detallado para el desarrollo del software.
- Proyecto: se debe planificar y controlar el proyecto para manejar su complejidad. Existen un conjunto de señales de advertencia que son comunes a todos los proyectos y es importante conocer, tanto como los factores de éxito claves que conducen a una buena administración de proyecto.

Los elementos y actividades clave en el proceso de gestión de proyectos son:

- Métricas
- Estimaciones
- Calendario temporal
- Organización del personal
- Análisis de riesgos
- Seguimiento y control

Planificación

La planificación es, en términos concisos, la prescripción de una secuencia operativa, que especifica:

- Qué debe hacerse
- Con qué recursos
- En qué orden

Si bien existen varios tipos adicionales, el curso se enfoca en dos tipos de planificación en particular.

Planificación organizativa

Vinculada a la gestión de recursos humanos (sean aquellos con los que ya se cuenta o aquellos con los que debería contarse). Como se mencionó anteriormente, el personal de una organización de software es su activo más grande, puesto que constituye su capital intelectual, es decir la herramienta principal con la que se llevan a cabo proyectos de software. Dado esto, resulta evidente que una mala administración del personal es uno de los factores principales para el fracaso de los proyectos.

Desde el punto de vista de los recursos humanos, uno de los aspectos a considerar es que hay una variedad de personas involucradas en un proyecto de software, que difieren tanto desde el punto de vista de sus capacidades y conocimientos, como intereses y niveles de involucramiento en el proyecto:

- Gerentes ejecutivos: definen temas empresariales
- Gerentes de proyecto: planifican, motivan, organiza y contratan a los profesionales
- Profesionales: aportan habilidades y conocimientos técnicos
- Clientes: especifican los requerimientos
- Usuarios finales: interactúan con el software

Líder de equipo

Un equipo de software debe organizarse de forma tal que maximice las capacidades de cada persona. Lograr esto es una de las tareas principales de un líder de equipo. Para definir los rasgos que una persona en este puesto debería tener, existen algunos modelos o guías:

Modelo MOI:

Motivación: un líder debe ser capaz de motivar al personal profesional para sacar el mayor provecho posible de su tiempo y conocimientos

Organización: debe contar contar con habilidades que le permitan modelar procesos y gestionarlos a fin de lograr los objetivos esperados

Incentivación de Ideas e Innovación: es importante dar lugar a los involucrados a la participación creativa en el proyecto, buscando y aceptando ideas nuevas y diferentes

Rasgos de un líder eficaz:

Capacidad de resolución de problemas, sean estos administrativos, técnicos, o de índole interpersonal. Un líder eficaz debería ser capaz de detectar a tiempo esta clase de dificultades, estructurar posibles soluciones y llevarlas a cabo, o identificar y motivar a las personas idóneas para desarrollar esas tareas, pudiendo siempre aplicar conocimientos aprendidos y criterios de eficiencia (reconocer cuando una vía de solución está resultando infructuosa y poder cambiar el rumbo).

Identidad administrativa: debe tener la confianza y presencia como para saber cuándo asumir el control de los procesos y actividades, y cuándo confiar en el saber experto del equipo profesional, reconociendo sus fortalezas y debilidades (las propias y ajenas).

Logro: debe premiar las acciones innovadoras y el esfuerzo adicional del equipo, para motivar al equipo a aportar nuevas ideas, así como demostrar que no se castiga al personal por correr riesgos de manera controlada.

Influencia y construcción del equipo: un líder de proyecto eficaz debe poder leer a la gente, reconociendo señales verbales y no verbales, y pudiendo actuar en consecuencia de esos mensajes. Es importante que pueda permanecer bajo control en situaciones de estrés, y pueda desescalar situaciones problemáticas dentro del equipo, logrando un clima amigable y colaborativo.

El equipo de software

Existen tantas formas de organizar un equipo de software como proyectos y organizaciones. Dado que cada proyecto tiene sus características propias, y cada organización su propia historia que informa sus decisiones de planificación. Sin embargo, existen una serie de factores que deben ser tenidos en cuenta al momento de plantear una estructura de equipo, y que pueden permitir realizar esta tarea de manera idónea para un proyecto determinado:

- Dificultad del problema a resolver: esto puede definir tanto el tamaño del equipo, como la experiencia necesaria (tanto en tiempo, como profundidad y variedad).
- Tamaño del programa resultante
- Tiempo que el equipo permanecerá unido
- Grado en que puede dividirse en módulos el problema a resolver
- Calidad y confiabilidad requerida por el sistema a construir
- Rigidez de la fecha de entrega
- Grado de sociabilidad o comunicación requerido para el proyecto

Comunicación grupal

Dado este último ítem , resulta importante mencionar que la comunicación en un grupo es siempre importante, ya que una comunicación fluida permite evitar problemas o detectarlos a tiempo, así como ayudar a mantener motivado al grupo al tener un conocimiento más acabado del estado del proyecto y tareas que puedan estarse llevando a cabo. La calidad de la comunicación en un grupo puede verse afectada por factores como el estatus de los miembros del equipo, el tamaño de éste, los canales de comunicación disponibles y características personales de sus miembros.

Los programadores pueden mejorar la productividad si cuentan con un entorno de trabajo provisto con recursos necesarios y áreas de comunicación adecuadas.

Organigramas de equipo

Existen tres organigramas genéricos de equipo. La elección del organigrama dependerá de las características del proyecto, su complejidad, el tiempo previsto para su desarrollo, e incluso en cierta medida de las características propias del personal involucrado. Sin embargo, se puede hacer una generalización sobre las situaciones para las que cada organigrama puede resultar más útil. En este caso, están ordenadas de mayor a menor complejidad de proyecto para el que son recomendados.

- Descentralizado democrático
  - No tiene un jefe permanente
  - Se nombran coordinadores de tareas a corto plazo
  - Las decisiones se toman por consenso
  - La comunicación entre los miembros del equipo es horizontal
- Descentralizado controlado
  - Tiene un jefe definido que coordina tareas específicas y jefes específicos para subtareas determinadas.
  - La resolución de problemas es una actividad del grupo, pero la implementación de soluciones
  - se reparte entre subgrupos.
  - Dentro del equipo existen tanto vías de comunicación vertical como horizontal.
- Centralizado controlado
  - El jefe del equipo se encarga de la resolución de problemas a alto nivel y de la coordinación interna del equipo.
  - La comunicación entre el jefe y los miembros del equipo es vertical.

Sugerencias

- Los proyectos muy grandes son mejor dirigidos por equipos con estructura CC o DC, donde se pueden formar fácilmente subgrupos.
- Es importante reconocer que el tiempo de convivencia, junto con su estilo de comunicación y jerarquía, puede afectar a la moral del equipo. En este sentido, los organigramas tipo DD son mejores para equipos que permanecerán juntos durante mucho tiempo.

Planificación temporal

Definición: actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto. Esta definición deja en claro que hay dos instancias de estimación interdependientes, de lo que resulta que ambas deben tratar de realizarse de la mejor forma posible a fin de evitar los problemas que puede traer aparejada una mala estimación de uno o ambos aspectos.

Calendarización

La definición de fecha final para el proyecto puede darse de dos formas posibles:

- Establecida por el cliente
- Establecida por los desarrolladores

En el último caso, puede optimizarse de mejor forma el uso de los recursos, ya que la fecha puede establecerse en función de éstos, y cómo los desarrolladores estipulan es la mejor forma de aprovecharlos. El primer caso, sin embargo, es el más frecuente, y en él se invierte el orden , ya que los recursos deben disponerse en función de la optimización del tiempo.La calendarización está compuesta por:

- Tareas: secuencia de acciones a realizar en un plazo determinado
  - Tarea crítica: es aquella cuyo retraso genera un retraso en todo el proyecto
- Hito: es un resultado que se espera sea obtenido en una determinada fecha. Generalmente se trata de resultados que surgen de la compleción de una tarea.

Parámetros de las tareas

- Precursor: evento o conjunto de eventos que deben ocurrir antes de que la actividad pueda comenzar
- Duración: cantidad de tiempo necesaria para completar la actividad
- Fecha de entrega: fecha para la cual la actividad debe estar completada
- Resultado: hito o componente listo

Red de tareas

Es una representación gráfica del flujo de las tareas desde el inicio hasta el fin de un proyecto. Se basa en la idea de que algunas tareas pueden realizarse en paralelo, mientras que otras son directamente dependientes de otras. La confección de la red permite visualizar claramente la secuencia e interdependencia de las tareas.

Gantt

Es otra forma de planificar visualmente todas las tareas. En este caso, se conforma una tabla en la que para cada tarea se marcan los lapsos de tiempo (fechas) en los que se debe o espera realizar. De esta forma, se puede visualizar fácilmente el paralelismo, así como las tareas que deberían estar activas en una determinada fecha en concreto del proyecto.

PERT (Program Evaluation and Review Techniques)

Es una metodología creada para la gestión de proyectos del programa de defensa del gobierno de EEUU entre 1958 y 1959. Su principal utilidad es en el control de proyectos con una gran cantidad de tareas o actividades que implican investigación, desarrollo y pruebas.

Se parece a la red de tareas, pero a cada una de ellas le asigna tiempos en tres categorías: probable, optimista y pesimista. Por esta característica, se lo considera esencialmente un método probabilístico antes que prescriptivo.

CPM (Critical Path Method)

Método desarrollado para DuPont (en ese momento líder en la producción de plásticos, que implementó un método precursor al CPM en el Proyecto Manhattan) y Remington (la empresa de máquinas de escribir, en ese momento productores de la UNIVAC) entre 1956 y 1958. Se emplea en proyectos en los que hay poca incertidumbre en las estimaciones. Establece tiempos específicos, aunque dispone tiempos de inicio tempranos y tardíos, por lo que se lo considera un método determinístico.

Método del Camino Crítico (PERT-CPM)

Actualmente se emplean estrategias planteadas en los métodos PERT y CPM en un solo método, denominado método del camino crítico (CPM por sus siglas en inglés, ya que hereda características claves de este). La metodología consiste en:

1. Establecer la lista de tareas que incluye el proyecto2. Establecer la duración de las tareas y sus relaciones de dependencia, es decir aquellas tareas que deben haber terminado para poder iniciar una tarea dada.
2. Construir la red
3. Numerar los nodos
4. Calcular la fecha temprana y tardía de cada nodo

T ei = fecha temprana (early) del nodo i

T ai = fecha tardía (absolute) del nodo i

Cálculo de la fecha temprana
$T_{eJ} = T_{eI} + t_{IJ}$
Donde
T_{eJ} = Fecha más temprana del nodo destino
T_{eI} = Fecha más temprana del nodo origen
t_{IJ} = Tiempo de la tarea desde el nodo I hasta el nodo J
En caso de haber más de un camino $T_{eJ} = \max(T_{eJ1} , T_{eJ2} , ...)$
Cálculo de la fecha tardía
Se estipula que la fecha tardía de todo el proyecto es igual a la fecha temprana de todo el proyecto. A partir de allí las fechas tardías se calculan desde el final hacia el principio.
$T_{aI} = T_{aJ} - t_{IJ}$
Donde
T_{aJ} = Fecha más tardía del nodo destino
T_{aI} = Fecha más tardía del nodo origen
t_{IJ} = Tiempo de la tarea desde el nodo I hasta el nodo J
En caso de haber más de un camino $T_{aJ} = \min(T_{aJ1} , T_{aJ2} , ...)$
6. Calcular el camino crítico que une las tareas críticas
$\Rightarrow T_{ei} = T_{ai}$
Cálculo del margen total
$Mt = T_{aJ} - T_{eI} - t_{IJ}$
Donde
T_{aJ} = Fecha más tardía del nodo destino
T_{eI} = Fecha más temprana del nodo origen
t_{IJ} = Tiempo de la tarea desde el nodo I hasta el nodo J
También puede calcularse como la diferencia entre la fecha más tardía y más temprana del mismo nodo.
Las tareas críticas son aquellas cuyo margen total es cero o próximo a cero, el camino crítico es aquel que une principio y fin con el menor margen total posible. Una tarea con un margen total de 6, entonces, es una que puede iniciarse con 6 unidades de tiempo (dependiendo la magnitud con que se planifique pueden ser días, horas, semanas, etc.) de retraso sin que ello afecte a la duración final del proyecto. En contrapartida, una tarea con margen total igual a cero, no puede retrasarse sin implicar un retraso en el proyecto en su totalidad.El PERT-CPM permite, luego de calcular todos los tiempos y márgenes, estipular las tareas y caminos críticos, así como definir la ventana temporal para cada actividad (es decir el margen de tiempo con el que puede iniciarse una tarea sin iniciar retrasos) y para el proyecto en su totalidad.
Tareas fuera de agenda
En el caso que el inicio de una tarea se retrase respecto a la planificación, lo primero que debe hacerse es revisar el impacto que éste retraso tiene sobre la fecha de entrega. En caso de tratarse de una tarea no crítica, es posible que el sólo hecho de iniciar la actividad sea suficiente para evitar retrasos. En caso contrario, será necesario considerar la reasignación de recursos. Esto último debe hacerse a través de un análisis riguroso de la situación que defina qué recursos y cómo se agregan para garantizar la finalización a término, puesto que, por ejemplo, incluir más personas a un proyecto en curso puede atentar negativamente a la productividad. Otras opciones en casos de retraso son el reordenamiento de tareas, si es que es posible lograrlo reduciendo o eliminando el retraso, o en última instancia la modificación de las fechas de entrega.

Métricas y estimaciones

Las métricas son una clave (o pista) tecnológica, que brinda las herramientas para entender, evaluar, controlar y por lo tanto mejorar los procesos de desarrollo (y mantenimiento) y sus productos resultantes.
Definiciones
Medida: indicación cuantitativa de una dimensión de un proceso o producto. Por ejemplo: cantidad de personas involucradas en todo un proceso, cantidad de líneas involucradas en un producto, tiempo necesario para el desarrollo del proyecto.
Medición: el proceso de obtener o estipular una medida. Una medición puede ser más o menos precisa dependiendo de lo que se está midiendo. Por ejemplo, establecer a priori el tiempo de desarrollo de un producto de software implica especulación (que puede tener mayor o menor grado de certidumbre según qué tan bien informada sea la inferencia), mientras que medir la cantidad tests que el producto no logra sobrepasar en un determinado momento involucra datos concretos y verificables.

Métrica: es la estrategia seleccionada para realizar una medición sobre un determinado proceso o producto (o una parte de cualquiera de ellos). Puede involucrar la realización de más de una medición y la agregación de las medidas resultantes. La temperatura, por ejemplo, es la base de una métrica, pero dependiendo del contexto puede ser más importante la temperatura ambiente, o la sensación térmica, estas últimas dos son efectivamente métricas.
Indicador: es una combinación de métricas que permite realizar una evaluación del estado del producto o proyecto en términos de su contexto. Por ejemplo, si los objetivos se están cumpliendo con una semana de retraso, es un indicador de que el proyecto está enfrentando dificultades de algún tipo (o que las estimaciones no fueron correctas).
Como se dijo con anterioridad, las métricas pueden ofrecer los medios (indicadores) para asegurar la calidad en procesos, proyectos y productos de software.

Métricas de proyecto

Sirven para medir efectivamente el estado del proyecto e informar decisiones respecto a éste:

- Ajustes en el calendario y prevención de demoras
- Valoración general del estado de un proyecto en un momento o intervalo de tiempo determinado
- Rastreo de riesgos
- Descubrimiento de problemas
- Ajuste del flujo de trabajo y/o tareas
- Evaluación de la habilidad del equipo

En cada etapa tarea de un proyecto, pueden (y deben) definirse e implementarse métricas para informar las decisiones que puedan involucrar la realización de los ajustes necesarios para la realización del proyecto según los criterios preestablecidos.

Métricas del proceso

Estas métricas suman valor en conjunto. La idea esencial es la recopilación: generación de una base de datos (en sentido amplio) que acumule las métricas de todos los proyectos en un intervalo de tiempo determinado. La intención de esta actividad es la de proporcionar un conjunto de indicadores que puedan servir para la toma de decisiones en proyectos en curso, y para mejorar los procesos a futuro (incluyendo las estimaciones involucradas en el proceso de gestión de un proyecto).
Etiqueta de métricas de proceso de software (Grady, en Pressman, 2010):

- Usar el sentido común y sensibilidad organizacional cuando se interpretan datos de métricas.
- Proporcionar retroalimentación regular a los individuos y equipos que recopilan medidas y métricas.
- No usar métricas para valorar a los individuos.
- Trabajar con los profesionales y con los equipos para establecer metas y métricas claras que se usarán para lograr las primeras.
- Nunca usar métricas para amenazar a los individuos o a los equipos.
- No considerar “negativos” los datos de métricas que indiquen un área problemática. Dichos datos simplemente son un indicio para mejorar el proceso.
- No obsesionarse con una sola métrica ni excluir otras métricas importantes.

Algunas características del proceso para las que puede resultar relevante definir métricas son:

- Comprensión: la medida en la que el proceso está definido explícitamente, y la facilidad con que esta definición puede ser comprendida
- Estandarización: la medida en que el proceso se basa en o se aproxima a un procesos genérico establecido por un estándar dado.
- Visibilidad: la facilidad con la que se pueden observar los resultados y mecanismos del proceso, depende también de la claridad con que éstos estén definidos.
- Mensurabilidad: la presencia en la definición propia del proceso de mecanismos de recolección de datos u otras actividades que permitan medir las características del proceso y/o producto.
- Soportabilidad: la medida en que se pueden utilizar herramientas de software para apoyar las actividades del proceso.
- Aceptabilidad: el nivel de aceptación que tiene el proceso para los responsables de elaborar el producto, y el nivel de utilidad que les provee el propio proceso para esta misma actividad (no es lo mismo que un proceso sea aceptado por quienes lo implementan a que les resulte útil).
- Fiabilidad: la medida en la que el proceso cuenta con herramientas o actividades que eviten o detecten tempranamente errores antes que deriven en errores de producto.
- Robustez: la capacidad del proceso a continuar a pesar de dificultades inesperadas.

- Mantenibilidad: la capacidad de evolución del proceso, para reflejar requerimientos cambiantes de la organización o mejoras identificadas en el proceso.
- Rapidez: la velocidad y facilidad con la que el proceso puede dar como resultado un producto a partir de una especificación dada.

Las métricas elaboradas para estas características sólo son útiles en tanto y en cuanto se analicen conforme a los objetivos que se tengan sobre el proceso en general. Es importante reconocer que muchas de estas características son sumamente interdependientes, y que la priorización de unas puede involucrar la depreciación de otras. Un incremento en la visibilidad, por ejemplo, probablemente conlleve un incremento en la comprensión, puesto que implicaría más puntos de definición clara del proyecto, pero esta correlación sólo sucede hasta cierto punto, en el que la cantidad de artefactos complejiza al proceso y por lo tanto su comprensión, pero también es posible que afecte la rapidez, dado que la generación de documentación y otros artefactos específicos consume más tiempo. Por otro lado, resulta también evidente que estas características son más cualitativas que cuantitativas, por lo que en muchas ocasiones su metrificación y evaluación estará fuertemente vinculada con la medición de características en el producto (por ejemplo la cantidad de líneas de código que se modifican en promedio por cada cambio en los requerimientos, o la cantidad promedio de funcionalidades aceptadas por el cliente luego de un determinado sprint - en el caso de metodologías ágiles).

Métricas del producto

Estas se subdividen generalmente en dos grandes subgrupos:

- Dinámicas: se realizan sobre un programa en ejecución. Ayudan a valorar la eficiencia y fiabilidad de un programa.
- Estáticas: se realizan sobre representaciones del sistema. Ayudan a valorar la complejidad, comprensibilidad y mantenibilidad.

Estáticas

Fan-in/Fan-out: Fan-in (abanico de entrada) es una medida del número de funciones o métodos que llaman a otra función o método (por ejemplo, X). Fan-out (abanico de salida) es el número de funciones a las que llama la función X. Un valor alto para fan-in significa que X está estrechamente acoplado con el resto del diseño y que los cambios a X tendrán extensos efectos dominó. Un valor alto de fan-out sugiere que la complejidad global de X puede ser alta debido a la complejidad de la lógica de control necesaria para coordinar los componentes llamados.

Longitud de código (LDC KLDC - líneas de código o miles de líneas de código respectivamente): Ésta es una medida del tamaño de un programa. Por lo general, cuanto más grande sea el tamaño del código de un componente, más probable será que el componente sea complejo y proclive a errores. Se ha demostrado que la longitud del código es una de las métricas más fiables para predecir la proclividad al error en los componentes.

Es una métrica relativamente frecuente y de larga data, sin embargo tiene la desventaja de sólo poder realizarse efectivamente una vez desarrollado el producto (lo que se llama métrica postmortem*). Esto le da una utilidad limitada, pero de todas formas puede servir como información comparativa en relación a otros productos, y para informar decisiones a futuro. Al comparar esta información con la de otros proyectos, se puede establecer una métrica de la calidad del proceso. Algunos de los aspectos que se pueden evaluar de esta forma son:

Productividad: KLDC/persona/mes

Calidad: errores/KLDC

Costo: $/KLDC

Vale aclarar que la medida puede no estar teniendo en cuenta la presencia de líneas de comentarios o en blanco. Por este motivo también se ha propuesto otra medida adicional que es la razón de líneas de comentarios (CLDC) respecto a las líneas totales, es decir CLDC/LDC.

**Las métricas postmortem permiten establecer una línea base para mediciones futuras. También generan un registro de información que permite definir estimaciones futuras, tanto con proyectos nuevos como con el seguimiento del proyecto del que surgen: el nivel de complejidad de un determinado producto (en la métrica que sea), permite establecer con cierto grado de certeza su mantenibilidad.*

Complejidad ciclomática: Ésta es una medida de la complejidad del control de un programa. Tal complejidad del control puede relacionarse con la comprensibilidad del programa. En el capítulo 8 se estudia la complejidad ciclomática.

Longitud de identificadores: Ésta es una medida de la longitud promedio de los identificadores (nombres para variables, clases, métodos, etcétera) en un programa. Cuanto más largos sean los identificadores, es más probable que sean significativos y, por ende, más comprensible será el programa.
Profundidad de anidado condicional: Ésta es una medida de la profundidad de anidado de los enunciados if en un programa. Los enunciados if profundamente anidados son difíciles de entender y proclives potencialmente a errores.
Índice Fog: Ésta es una medida de la longitud promedio de las palabras y oraciones en los documentos. Cuanto más alto sea el valor del índice Fog de un documento, más difícil será entender el documento.
Generación de métricas
Desarrollo de una métrica-GQM: Dicho método esta orientado a lograr una métrica que "mida" cierto objetivo. El mismo nos permite mejorar la calidad de nuestro proyecto.
Estructura :
-Nivel Conceptual (Goal / Objetivo): Se define un objetivo (en nuestro caso, para el proyecto).
-Nivel Operativo (Question / Pregunta): Se refina un conjunto de preguntas a partir del objetivo, con el propósito de verificar su cumplimiento.
-Nivel Cuantitativo (Metric / Métrica): Se asocia un conjunto de métricas para cada pregunta, de modo de responder a cada una de un modo cuantitativo.
GQM-(OPM)
Es útil para decidir qué medir.
Debe estar orientado a metas.
Es flexible.
Estimaciones
Mientras las métricas implican la medición de datos concretos, las estimaciones implican necesariamente un grado de incertidumbre en su disposición, puesto que no pueden basarse más que en inferencias a partir de información incompleta o intuición basada en la experiencia.
Es importante reconocer la naturaleza incierta de las estimaciones. Sin embargo, dicha falta de certeza puede ser reducida si se hace un esfuerzo particular al momento de realizar una estimación, por ejemplo empleando varias fórmulas de estimación diferentes, y generando una estimación a partir de una media o promedio entre los valores obtenidos, o comparando con los datos históricos de proyectos o procesos similares.
En general, buena parte de los requerimientos de un proyecto (recursos, tiempos, etc.) deben estimarse al menos en sus fases iniciales. En la medida en que un proyecto avanza, es más factible establecer con mayor grado de certeza o incluso medir estos datos con base en las actividades ya realizadas y las entradas y salidas de éstas.
Estimaciones de recursos
Prueba de software
Este proceso se desarrolla en conjunto con el propio software, y no es sólo el hecho puntual de poner a prueba el software, sino también todos los puntos del proyecto en los que se definen características esperadas o criterios de validación o aceptación de dichas características (por ejemplo al definir los criterios de aceptación al escribir las HU). A su vez, podría decirse que, una vez en producción (es decir ya entregado y 'finalizado' el desarrollo), el software se encuentra en un estado de prueba constante.
Es importante destacar que se considera que todo software tiene errores. La finalidad de las pruebas no es necesariamente brindar información suficiente para eliminarlos a todos, si no la que sea necesaria para minimizarlos a valores tolerables. Cuando un conjunto de pruebas no produce fallos, lo único que permite garantizar es que el software no falla sólo bajo las condiciones que implican las pruebas (es decir que no garantizan la ausencia general de fallos), por eso es importante el correcto diseño de las mismas.
Falla de software
Una falla es una situación en la cual el software no cumple con uno o más de los requerimientos especificados. En este sentido el software puede no tener ningún tipo de error técnico, e incluso cumplir con lo que se espera de él, y aún así considerarse un fallo, por ejemplo si los requerimientos están especificados de forma incorrecta (es decir que no reflejan la realidad de lo que se espera del software). Otras condiciones o causas de fallos pueden ser:

- Requerimientos imposibles de cumplir con las estructuras previstas
- Defectos en el diseño del sistema
- Defectos en el diseño del programa
- Defectos en el código

De forma más general, pueden establecerse una serie de categorías a las cuales adscribir diferentes condiciones de fallo:

- Defecto algorítmico: se refiere a los problemas que pueden surgir de un código con errores de base, como puede ser la no inicialización de variables, o no chequear por condiciones que puedan llevar a bucles infinitos.
- Defecto sintáctico: son fallos que derivan de una mala interpretación y/o escritura del código, como por ejemplo variables o palabras clave referenciadas con nombres incorrectos, confusión de tipos (O y 0 - o mayúscula y cero - I y l - i mayúscula y l minúscula)
- Defecto de precisión: el origen del problema está en la mala o incorrecta implementación de fórmulas o algoritmos, resultando en errores de precisión en los resultados que pueden o no propagarse. Puede darse como ejemplo la precisión en la expresión de decimales (o incluso el tipo que se utiliza) en software de contabilidad, o el tamaño de una hitbox en un juego tipo FPS)
- Defecto de documentación: este caso se refiere a casos como el ejemplo dado anteriormente (aunque no exclusivamente), en los que el problema surge esencialmente de una documentación que no es acorde con lo que el software hace efectivamente. En este caso puede suceder tanto que el software produzca los resultados esperados por la documentación, pero que por ser esta incorrecta ambos aspectos deban rehacerse, o que el software cumpla con los resultados esperados, pero no con la especificación, en cuyo caso sólo debería revisarse la documentación.
- Defecto de sobrecarga: el software funciona, pero no cumple con los requisitos de rendimiento
- esperados, por ejemplo si colapsa a partir de determinado umbral de usuarios conectados (siempre que ese umbral esté dentro de los requerimientos documentados).
- Defecto de capacidad: el sistema no admite la cantidad de datos que se espera según los requerimientos, por ejemplo el caso de una base de datos muy pequeña para un sistema con un gran flujo de altas.
- Defecto de coordinación o sincronización: se refiere específicamente a problemas de concurrencia y/ o coordinación entre procesos. Puede ser por ejemplo el caso de un deadlock, o que una compra sea ejecutada antes de que concluya la validación del medio de pago.
- Defecto de recuperación: identifica a las situaciones en las cuales el sistema no es capaz de recuperarse de un fallo.
- Defecto de relación hardware-software: se refiere a la incompatibilidad entre componentes de hardware, o de éstos con el propio software.
- Defecto de estándares: el caso en el que el software funciona, pero no alcanza a cumplir con los estándares y procedimientos definidos en la planificación.

Aún así, una falla puede adscribirse a más de una de estas categorías, y los límites entre una y otra pueden resultar difusos. Por este motivo, existen otras formas de clasificar fallas.
Una aproximación inicial es la de clasificarlas a partir del tipo de defecto en dos grandes categorías:

- Defecto de omisión: en este caso el problema está en la falta de un elemento o componente (un comportamiento no está descrito en la especificación, una variable no está inicializada, etc.)
- Defecto de cometido: la falla no es ocasionada por una falta de información, si no por una manipulación incorrecta de la misma (se emplea un algoritmo ineficiente, se realiza una operación inválida sobre un tipo de dato dado)

Otra forma de categorizar, algo más exhaustiva, es la clasificación ortogonal de defectos, que establece 8 categorías posibles a las cuales puede pertenecer una falla:

- Función: afecta a la capacidad o interfaces
- Interfaz: afecta a la interacción con otros componentes
- Comprobación: afecta a la lógica del programa
- Asignación: afecta a la estructura de datos
- Sincronización: involucra la sincronización de recursos compartidos y de tiempo real
- Construcción: ocurre debido a problemas en repositorios, gestión de cambios o control de versiones
- Documentación: afecta a publicaciones
- Algoritmo: involucra la eficiencia o exactitud de un algoritmo

Prueba de software

El objetivo elemental de este proceso es sacar a la luz todos los errores (fallos) que existan en el producto. En tanto estrategia, el foco central está en el diseño de pruebas que puedan lograr este objetivo con la menor inversión de tiempo y esfuerzo posible. Se considera exitosa a una prueba lleva al descubrimiento de un

error. En caso contrario, se trata de una prueba mal diseñada (no pone al sistema en condiciones de error), o directamente de una prueba 'inútil' (implicando que se invirtió tiempo y esfuerzo en detectar un error inexistente)

Beneficios

El desarrollo de pruebas de software permite:

- Descubrir errores antes de que el software salga del ambiente de desarrollo
- Detectar errores no descubiertos hasta el momento
- Mientras mayor sea el grado en que los puntos anteriores se cumplan, menor será el costo de la corrección de errores en la etapa de mantenimiento

Principios

Como se mencionó anteriormente, es importante planificar las pruebas mucho antes de que empiecen, o incluso de que sean susceptibles de empezar (es decir antes de que las funciones que se comprueban sean efectivamente codificadas). En este sentido, a su vez, para cualquier prueba debería encontrarse un camino de información que dirija directamente a uno o más requisitos del cliente, lo que implica por otro lado que no deberían existir pruebas ni condiciones de error que no estén ligados con algún requisito del cliente.

En cuanto a la distribución del esfuerzo, puede aplicarse el principio de Pareto, según el cual el 20% del código es responsable del 80% de los errores, mientras que el 80% del código restante es responsable de tan sólo el 20% de fallas. Con esta información, resulta evidente la importancia de reconocer las funciones críticas o puntos problemáticos principales, ya que abocando esfuerzo a la solución de los problemas que allí surjan, se podrá reducir drásticamente la cantidad de fallas en el software.

Estrategia

Es recomendable que las pruebas vayan decrementando en granularidad a medida que progresan, es decir que vayan 'de lo particular a lo general'. En este camino, es importante asegurarse que se hayan probado todas las condiciones a nivel de componente.

Por último, es también recomendable que las pruebas sean realizadas por un equipo independiente, que no tenga compromisos ni prejuicios sobre el código ya producido y su funcionamiento.

Tipos de prueba

Caja blanca / caja de cristal / caja abierta

Se basan en el examen minucioso de detalles procedimentales, accediendo a diferentes módulos del código y poniéndolos a prueba de manera puntual, a través de casos de prueba que los involucren directamente.

Los casos de prueba se derivan de la estructura de control del programa, por lo que debe garantizarse que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.

Un punto importante a tener en cuenta al momento de realizar estas pruebas, es que el flujo lógico de un programa puede resultar muy diferente al que se intuye al momento de revisarlo o siquiera de codificarlo. Por este motivo, pueden existir múltiples errores de diseño que no serán percibidos hasta el momento de realizar las pruebas que pasen por esos caminos de fallo. Respecto a esto último, es importante recalcar que un camino de fallo no es sólo el flujo lógico que llevó al mismo, si no también el conjunto de datos o el contexto general de ejecución que lo hicieron posible.

Prueba del camino básico

Permite al diseñador de pruebas obtener una medida -denominada complejidad ciclomática- de la complejidad lógica del programa, y usarla como guía para la definición de caminos de ejecución. Los casos de prueba obtenidos por este medio, garantizan que se ejecuta al menos una vez cada sentencia del programa.

Se inicia con la definición del flujo en forma de grafo, con lo que las estructuras de control más comunes se representarían de la siguiente forma:

Cada nodo en el grafo de flujo representa una o más sentencias procedimentales. Cada nodo que incluya una condición es denominado nodo predicado, y se caracteriza porque dos o más aristas surgen de él. A su vez, al área encerrada entre aristas se la conoce como región. En este sentido, el área ocupada por el grafo completo es conocida como región total, y la misma puede ser abierta o cerrada.

La complejidad ciclomática es el número de caminos independientes que se pueden realizar, es decir la cantidad de caminos que pueden hacerse que introduzcan al menos un nuevo conjunto de sentencias de proceso o una nueva condición, respecto a cualquier otro camino. Este valor provee un límite superior para el número de pruebas que se deben ejecutar si se busca garantizar que se ejecutan todas las sentencias al menos una vez.

La complejidad ciclomática se calcula como:

- V(g) = La cantidad de regiones del grafo

- V(g) = A - N + 2 (A: aristas, N: nodos)
- V(g) = P + 1 (P: predicados)

Debe calcularse con las tres fórmulas, para garantizar su exactitud.
Los pasos involucrados en la realización de la prueba del camino básico son, entonces:

1. Dibujar el grafo
2. Determinar la complejidad ciclomática
3. Determinar un conjunto básico de caminos independientes
4. Preparar los casos de prueba que forzarán la ejecución de cada camino del conjunto
5. Ejecutar cada caso de prueba y comparar los resultados obtenidos con los esperados

Caja negra / caja cerrada / de comportamiento
Ponen a prueba el software a partir de las interfaces que éste provee. Las pruebas se diseña de manera agnósitca a las particularidades de codificación del producto. En este sentido, se las puede considerar análogas al concepto de objeto en POO, dado que se interesan específicamente en los ‘mensajes’ que el software puede responder (las formas de interacción con el entorno que provee), y no en la forma en que estos se implementan.
Se utilizan para buscar errores del tipo:

- Ausencia o fallo de funciones
- Errores de interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

Prueba de partición equivalente
El diseño de los casos de prueba se basa en una evaluación de las clases de equivalencia para una condición de entrada:
Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada.
Una condición de entrada es:

- Un valor numérico específico: para esta condición se establecen tres clases de equivalencia, una válida: el valor numérico válido, y dos no válidas: los rangos de valores por fuera de ese número.
- Un rango de valores: este caso es igual al anterior, con la diferencia que la clase de equivalencia válida es todo el rango de valores válidos.
- Un conjunto de valores relacionados: aquí se definen dos clases de equivalencia, una válida: elementos pertenecientes al conjunto válido, y una inválida: elementos no pertenecientes al conjunto.
- Una condición lógica: este caso es fácilmente comparable al anterior, ya que una condición lógica no es más que un conjunto de valores de verdad. De esta forma se tienen también dos clases de equivalencia.

Análisis de valores límite (AVL)
Parte de la base que los errores tienden a acumularse más en los extremos de las condiciones de entrada que en el centro (esta idea es más clara cuando se habla de rangos, pero es aplicable a todos los casos). Esta disciplina complementa a la partición equivalente, ya que en lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL selecciona los casos de prueba en los ‘extremos’ de los valores posibles (o, mejor dicho, en los márgenes entre los estados válidos y los no válidos). A su vez, el AVL no se concentra exclusivamente en las condiciones de entrada, si no que también obtiene casos de prueba para el campo de salida.
Mantenimiento
Independientemente del modelo de proceso, ésta es la última etapa del ciclo de vida del software. Las tareas aquí efectuadas pueden ser varias, dependiendo del objetivo concreto, pero en general se puede definir al mantenimiento como una etapa en la que se atiende al software y a su evolución una vez entregado este. Como por lo general implica modificaciones, adaptaciones y mejoras - muchas veces respondiendo no sólo a problemas en el software, si no a la evolución de los propios requerimientos del cliente - este proceso también es conocido como evolución del sistema.
Un equipo que realiza mantenimiento de un software no necesariamente es el mismo que lo ha desarrollado. En muchas ocasiones, se trata de hacer mantenimiento a sistemas heredados (sistemas legacy, aquellos que han quedado desactualizados, pero cuya actualización o modificación es costosa o indeseable). En estos casos, y con frecuencia también en software algo más reciente, el equipo de mantenimiento se encuentra con

un sistema anticuado en muchos aspectos, desde la interfaz a las soluciones algorítmicas, desarrollado sin una metodología o disciplina clara, sin documentación ni modularidad.
Situaciones como ésta, a veces incluso con una complejidad creciente de problemas a resolver, llevan a considerar que determinados contextos pueden involucrar la necesidad de directamente cerrar el ciclo de vida del sistema en mantenimiento, y comenzar otro para un sistema que lo reemplace. En estos casos, la economía es el punto decisivo: deben ponerse en la balanza los costos de mantenimiento del software actual contra contra los que pueda implicar el nuevo desarrollo y el proceso de reemplazo. Dado que el mantenimiento involucra no sólo la solución de errores, si no también actividades como agregado de mejoras, optimización, actualización o compatibilización con nuevas tecnologías, y otro tipo de modificaciones incrementales. Dependiendo de la envergadura de las tareas a realizar, el costo adicional puede resultar muy elevado, especialmente si se lo compara con los beneficios obtenidos. El punto a partir del cual resulta más redituable la construcción de un sistema nuevo que el mantenimiento del actual, es conocido como la barrera
de mantenimiento.

Consideraciones

- La avocación de esfuerzos al mantenimiento implica necesariamente que disminuye el personal disponible para otros desarrollos.
- Las modificaciones que surjan durante esta etapa pueden producir efectos secundarios indeseados, y a veces no percibidos hasta bastante más tarde, sobre el código, datos o incluso sobre la
- documentación. De esta forma, es posible que en un intento por incrementar la calidad del producto, ésta termine reduciéndose en su lugar.
- Las tareas de mantenimiento no sólo implican la modificación de implementaciones, si no que muchas veces para lograr esto deben reiniciarse las fases de análisis, diseño e implementación.

Con todo esto, resulta evidente que el mantenimiento no es una tarea sencilla ni barata. De hecho se calcula que esta etapa puede involucrar entre un 40% y un 70% del costo total de desarrollo.
Otro punto no menor a considerar, es que los errores devenidos de esta etapa, provocan la insatisfacción del cliente, lo cual funciona en detrimento de la imagen del equipo y/o empresa encargada del mantenimiento.

Problemas

Trabajar sobre un sistema ya desarrollado, implica varias cuestiones que hacen que la tarea de mantenimiento resulte poco atractiva:

- Hay pocas oportunidades para el desarrollo creativo, ya que se está trabajando en los límites que impone el sistema.
- Durante el diseño no siempre se prevén los cambios, con lo que las modificaciones pueden resultar extremadamente complejas.
- Puede resultar difícil comprender el código desarrollado por otros, más aún ante la falta de documentación o con documentación inadecuada.

Actividades

Al igual que en la gestión de configuración de software (de hecho, el mantenimiento implica precisamente gestión de configuración), deben emplearse mecanismos que permitan identificar, controlar e implementar los cambios que se produzcan en esta etapa.
Como puede deducirse de los apartados anteriores, el proceso de cambio se ve muy facilitado si durante el desarrollo están presentes atributos de calidad como modularidad, documentación interna del código fuente y documentación de apoyo.
Ciclo de mantenimiento

- Análisis: compresión del alcance y efecto de la modificación.
- Diseño: generar un rediseño que incluya la incorporación de los cambios.
- Implementación: recodificación y actualización de la documentación interna del código.
- Prueba: revalidación del software.
- Actualización de la documentación de apoyo.
- Distribución e instalación de nuevas versiones.

Desarrollo y mantenimiento

El mantenimiento no es una etapa que existe por sí sola, si no que es extremadamente dependiente de las decisiones y acciones que se hayan llevado a lo largo del desarrollo. Durante las etapas de creación del sistema, pueden tomarse ciertas consideraciones que facilitarán luego el mantenimiento, convirtiéndolo en una tarea más sencilla y menos costosa:

- Análisis: señalar principios generales, armar planes temporales, especificar controles de calidad, identificar posibles mejoras, estimar recursos para el mantenimiento.
- Diseño arquitectónico: claro, modular, modificable, con notaciones estandarizadas
- Diseño detallado: notaciones para algoritmos y estructuras de datos, especificación de interfaces,
- manejo de excepciones, efectos colaterales
- Implementación: indentación, comentarios de prólogo e internos, codificación simple y clara
- Verificación: lotes de prueba y resultados

Tipos de mantenimiento

- Correctivo: diagnóstico y corrección de errores. Esto no sólo se refiere a errores en el propio código, si no que abarca todos los errores que puedan existir en el software y componentes asociados, como pueden ser problemas de rendimiento, o incluso en la documentación.
- Adaptativo: se produce por la aparición de cambios en el entorno en el que funciona el sistema, que requiere una adaptación del sistema a dicho contexto. Puede tratarse de cuestiones de hardware o de software, desde características del almacenamiento al tipo y estructuras de datos, así como las interfaces de usuario requeridas.
- Perfectivo: el objetivo es mejorar el sistema, sea agregando una funcionalidad deseada, perfeccionando una ya existente, o mejorando las características de rendimiento, seguridad u otros requerimientos no funcionales.
- Preventivo: se efectúa antes de que surja una petición que lo requiera, para facilitar el futuro mantenimiento. Se basa en el conocimiento preexistente sobre el producto para identificar las posibles modificaciones. En general no se trata de cambios que alteren en gran medida las funcionalidades, pero pueden involucrar desde una reorganización del código hasta mejoras en el rendimiento, portabilidad, seguridad, etc. del sistema.

Rejuvenecimiento del software

Es un desafío del mantenimiento, que consiste en buscar el incremento de la calidad global de un sistema existente, previniendo futuros errores e incrementando la vida útil del producto, retrasando su degradación. Contempla retrospectivamente los subproductos de un sistema para intentar derivar la información adicional o reformarlo de un modo comprensible. Existen cuatro tipos o enfoques de rejuvenecimiento.

- Re-documentación: consiste en realizar un análisis del código para generar o mejorar la documentación del sistema. Puede incluir análisis del flujo de datos, diagramas de la jerarquía de componentes, caminos de prueba, documentación del código, etc.
- Re-estructuración: se ocupa de simplificar la estructura del código, de reordenarlo y estructurarlo. Puede involucrar la generación de documentación que exprese el resultado final, pero eso no es parte intrínseca de este proceso.
- Ingeniería inversa: en casos en los que sólo se cuenta con el código fuente, se parte de éste para generar toda la documentación del proyecto, llegando tal vez hasta la especificación, a través de inferencia.
- Re-ingeniería: es una extensión de la ingeniería inversa. Luego de producida la documentación en la ingeniería inversa, se busca re-estructurar el código fuente, mejorando su calidad sin intervenir en la funcionalidad del sistema.

Auditoría

En general, una auditoría es un examen crítico que se realiza con el objetivo de evaluar la eficiencia y eficacia de una entidad determinada (un organismo o una parte de él, un proceso, etc.) y determinar cursos alternativos de acción para mejorarla en pos de los objetivos propuestos.

Es una actividad con una alta carga de análisis, y que depende en gran medida de quien la realiza. Se espera que un auditor pueda ser objetivo, y que a su vez tenga un juicio profesional sólido y maduro, que le permitan evaluar la total complejidad del sistema que está siendo auditado.

Una auditoría puede ser de carácter interno, externo, o una combinación de ambas. Una auditoría interna es realizada por un agente que pertenece a la organización que está siendo auditada, en este caso, se compromete parte de la objetividad en pos de la ventaja que puede ofrecer tener mayor familiaridad con ciertas características del sistema a evaluar. Una auditoría externa, por otro lado, es realizada por un agente externo al sistema a auditar; en este caso se puede contar con una mayor objetividad del auditor, a costa de tal vez perder un poco de profundidad en la auditoría. Por lo general se recomienda realizar auditorías de ambos tipos, a fin de obtener y compensar las ventajas y desventajas de ambas.

Auditoría informática

En el contexto más reducido de la informática, por auditoría se entiende a la revisión y evaluación de:

- Controles, sistemas y procedimientos de la informática.
- Equipos de cómputo.
- La organización que participa en el procesamiento de información.

Se trata de una actividad preventiva, en la que el auditor revela problemas y sugiere soluciones. En general se trata de la prevención de delitos, problemas legales, técnicos, normativos, etc.
Los procedimientos de auditoría pueden variar de acuerdo con la filosofía y técnica de cada organización y departamento de auditoría particular. De todas formas, la evaluación debe abarcar todas las aristas posibles del sistema.
Objetivos
Según Ron Weber: "Es una función que ha sido desarrollada para asegurar la salvaguarda de los activos de los sistemas de computadoras, mantener la integridad de los datos, y lograr los objetivos de la organización en forma eficaz y eficiente"
Según William Mair: " Es la verificación de los controles en las siguientes tres áreas de la organización [informática]: aplicaciones, desarrollo de sistemas, instalación del centro de cómputos."
A partir de estas definiciones, es posible reconocer los objetivos principales de la auditoría informática:

- Salvaguardar los activos: garantizar la perdurabilidad de los bienes (materiales e inmateriales) de la organización y sus sistemas.
- Garantizar la integridad de datos (en concordancia con el punto anterior)
- Asegurar la efectividad de los sistemas: buscar que los sistemas efectivamente resuelvan los problemas para los que fueron pensados o, en otras palabras, que cumplan su función.
- Asegurar la eficiencia de los sistemas: apuntar que los objetivos de los sistemas sean alcanzados con el menor uso de recursos posible.
- Mantener y mejorar la seguridad y confidencialidad de los sistemas y sus datos.

Campo de acción

- Evaluación administrativa del área informática
- Evaluación de los sistemas y procedimientos, y de la eficiencia que se tiene en el uso de la información
- Evaluación del proceso de datos, de los sistemas y de los equipos de cómputo (software, hardware, redes, bases de datos, comunicaciones)
- Seguridad y confidencialidad
- Aspectos legales de los sistemas y de la información

Diseño de la interfaz del usuario
El proceso de análisis y diseño de interfaces de usuario es iterativo.
Tareas:

- Análisis y modelado: Definir objetos y acciones de la interfaz (operaciones) con el uso de la información desarrollada en el análisis de la interfaz.
- Diseño de la interfaz: Definir eventos (acciones del usuario) que harán que cambie el estado de la interfaz de usuario. Hay que modelar este comportamiento.
- Construcción de la interfaz: Ilustrar cada estado de la interfaz como lo vería en la realidad el usuario final.
- Validación: Indicar cómo interpreta el usuario el estado del sistema a partir de la información provista a través de la interfaz.

Diseño de Interfaces y el diseño de experiencias de usuario (Udx)
El diseño de experiencias de usuario (Udx) es un conjunto de métodos aplicados al proceso de diseño que buscan satisfacer las necesidades del cliente y proporciona una buena experiencia a los usuarios destinatarios.
Se toma información de diferentes fuentes que permiten estudiar al usuario:
Encuestas, información de ventas, información de mercadotécnia, información de charlas de apoyo al usuario.
Reglas básicas del diseño
Dar control al usuario: El usuario busca un sistema que reaccione a sus necesidades y lo ayude a hacer sus tareas.
Reducir la carga de memoria del usuario: Reducir la demanda a corto plazo, definir valores por defecto que tengan significado,

definir accesos directos intuitivos.
Lograr una interfaz consistente: Permitir que el usuario incluya la tarea actual en un contexto que tenga algún significado, mantener consistencia en toda la familia de aplicaciones,mantener modelos que son prácticos para el usuario, a menos que sea imprescindible cambiarlos.
Factores Humanos: percepción visual/auditiva/táctil, memoria humana, razonamiento, capacitación Comportamiento/Habilidad personales, diversidad de usuarios.

Principios de Nielsen

Son una serie de principios que enuncian el dialogo correcto que debe proveer una interfaz con el usuario.

1. Dialogo simple y natural
2. Lenguaje familiar para el usuario
3. Evitar que el usuario esfuerce su memoria para interactuar con el sistema.
4. Que no haya ambigüedades con el aspecto visual, tecnológicos o de comportamiento.
5. Feedback, respuestas en pantalla frente a acciones del usuario.
6. Salidas evidentes, el usuario debe tener a su alcance una opción de salida,
7. Mensajes de error, feedback del sistema ante un error para ayudar a salir del mismo.
8. Prevención de errores, evitando que el usuario llego a un error.
9. Atajos para acciones que resulte cómodo y sencillo para usuarios expertos e inexpertos.
10. Ayudas, presencia de componentes de asistencia para el usuario.