Resumen de Ing. De Software

Unidad 1: Introducción a la Ingeniería de Software

Introducción a la Ingeniería del Software, ¿Qué es?

La Ingeniería del software es una disciplina de la ingeniería que comprende todos los aspectos de la producción del software, desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de este después de que utiliza.
En general los ingenieros de software adoptan un enfoque sistemático y organizado en su trabajo, ya que es la forma más efectiva de producir software de alta calidad.

Proceso de software
Un proceso de software es un conjunto de actividades y resultados asociados que producen un producto de soltare. Estas actividades son llevadas a cabo por los ingenieros de software.
Existen 4 actividades fundamentales de procesos, que son comunes para todos los procesos de software:

- Especificación del software, donde los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.
- Desarrollo del software, donde el software se diseña y programa.
- Validación del software, donde el software se valida para asegurar que es lo que el cliente requiere.
- Evolución del Software, donde el software se modifica adaptarlo a los cambios requeridos por el cliente.

La crisis del software
Actualmente casi todos los países dependen de complejos sistemas informáticos, casi todos los artefactos de uso diario están asociados a un programa de computadora, por lo que es fundamental producir software costeable y de calidad.
La crisis del software, fue el resultado de la introducción de las nuevas computadoras hardware basadas en circuitos integrados. Su poder hizo que las aplicaciones hasta ese entonces irrealizables fueran una propuesta factible. El software resultante fue de ordenes de magnitud mas grande y mas complejos que los sistemas de software previos.
La experiencia previa en la construcción de estos sistemas mostró que un enfoque informal para el desarrollo del software no era muy bueno. Los grandes proyectos a menudo tenían años de retraso, costaban mucho mas de los presupuestado, eran irrealizables, difíciles de mantener y con desempeño pobre.
El desarrollo del software estaba en crisis, se necesitaban nuevas técnicas y métodos para controlar la complejidad inherente a los sistemas grandes.
Estas técnicas han llegado a ser parte de la ingeniería del software y son ampliamente utilizadas. Sin embargo, cuanto mas crezca nuestra capacidad para producir software, también lo hará la complejidad de los sistemas de software asociados.
Al día de hoy, todavía existen muchas compañías que no aplican correctamente las técnicas de la ingeniería del software y es por eso que tenemos muchos proyectos que terminan en la nada.

Algunos "síntomas" que indican que el software se encuentra en un periodo de crisis son:

- Baja Calidad del Software.
- Tiempo y Presupuesto Excedido.
- Confiabilidad Cuestionable.
- Altos Requerimientos de Personal para desarrollo y mantenimiento.

Y algunas posibles causas de entrar en crisis son:

- Proyectos gestionados con un sobre-presupuesto.
- Proyectos gestionados con sobre tiempo.
- Software de baja calidad.
- El software a menudo no satisfacía los requerimientos deseados.
- Los proyectos fueron inmanejables, con un código difícil de mantener.

Ciclos de vida y su influencia de la Adm. De Proyectos/Ventajas y desventajas

La ingeniería de software tiene varios modelos o paradigmas de desarrollo en los cuales se puede apoyar para la realización de software.
Los ciclos de vida sirven para proveer una guía para la administración de proyecto y nos ayudan a responder las siguientes preguntas:

- ¿Cuales de las tareas deben ser rastreadas?
- ¿Que tipo de progreso se ha realizado?

Desarrollo en cascada
En Ingeniería de software el desarrollo en cascada, es el enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma tal que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.
De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costes del desarrollo.
Ventajas:

- Se tiene todo bien organizado y no se mezclan las fases.
- Es perfecto para proyectos que son rígidos, y además donde se especifiquen muy bien los requerimientos y se conozca muy bien la herramienta a utilizar

Desventajas:

- En la vida real, un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.
- Difícilmente un cliente va a establecer al principio todos los requerimientos necesarios, por lo que provoca un gran atraso trabajando en este modelo, ya que este es muy restrictivo y no permite movilizarse entre fases.
- Los resultados y/o mejoras no son visibles, el producto se ve recién cuando este esté finalizado, lo cual provoca una gran inseguridad por parte del cliente que anda ansioso de ver avances en el producto. Esto también implica toparse con requerimientos que no se habian tomado en cuenta, y que surgieron al momento de la implementación, lo cual provocara que se regrese nuevamente a la fase de requerimientos.

Desarrollo en espira
Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.
Básicamente consiste en una serie de ciclos que se repiten en forma de espiral, comenzando desde el centro. Se suele interpretar como que dentro de cada ciclo de la espiral se sigue un Modelo Cascada, pero no necesariamente debe ser así.
Ventajas:
El análisis del riesgo se hace de forma explícita y clara. Une los mejores elementos de los restantes modelos.

- Reduce riesgos del proyecto
- Incorpora objetivos de calidad
- Integra el desarrollo con el mantenimiento, etc.

Además es posible tener en cuenta mejoras y nuevos requerimientos sin romper con la metodología, ya que este ciclo de vida no es rígido ni estático.
Desventajas:

- Genera mucho tiempo en el desarrollo del sistema
- Modelo costoso
- Requiere experiencia en la identificación de riesgos

Modelo de prototipos
En Ingeniería de software el modelo de prototipos que pertenece a los modelos de desarrollo evolutivo, se inicia con la definición de los objetivos globales para el software, luego se identifican los requisitos conocidos y las áreas del esquema en donde es necesaria más definición. Entonces se plantea con rapidez una iteración de construcción de prototipos y se presenta el modelado (en forma de un diseño rápido). El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario. El diseño rápido conduce a la construcción de un prototipo, el cual es evaluado por el cliente o el usuario para una retroalimentación; gracias a ésta se refinan los requisitos del software que se desarrollará. La iteración ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.
Ventajas:

- Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
- También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina.
- El proceso procede linealmente (con poca información se retroalimenta)
- Resalta los riegos de la tecnología con el prototipo.

Desventajas:

- El usuario tiende a crearse unas expectativas cuando ve el prototipo de cara al sistema final. A causa de la intención de crear un prototipo de forma rápida, se suelen desatender aspectos importantes, tales como la calidad y el mantenimiento a largo plazo, lo que obliga en la mayor parte de los casos a

reconstruirlo una vez que el prototipo ha cumplido su función. Es frecuente que el usuario se muestre reacio a ello y pida que sobre ese prototipo se construya el sistema final, lo que lo convertiría en un prototipo evolutivo, pero partiendo de un estado poco recomendado.

- En aras de desarrollar rápidamente el prototipo, el desarrollador suele tomar algunas decisiones de implementación poco convenientes (por ejemplo, elegir un lenguaje de programación incorrecto porque proporcione un desarrollo más rápido). Con el paso del tiempo, el desarrollador puede olvidarse de la razón que le llevó a tomar tales decisiones, con lo que se corre el riesgo de que dichas elecciones pasen a formar parte del sistema final.

Desarrollo por etapas
El modelo de desarrollo de software por etapas es similar al Modelo de prototipos ya que se muestra al cliente el software en diferentes estados sucesivos de desarrollo, se diferencia en que las especificaciones no son conocidas en detalle al inicio del proyecto y por tanto se van desarrollando simultáneamente con las diferentes versiones del código.
Pueden distinguirse las siguientes fases:

- Especificación conceptual
- Análisis de requerimientos
- Diseño inicial
- Diseño detallado, codificación, depuración y liberación

Estas diferentes fases se van repitiendo en cada etapa del diseño.

Desarrollo iterativo incremental
Desarrollo iterativo y incremental es un proceso de desarrollo de software, creado en respuesta a las debilidades del modelo tradicional de cascada.
La idea principal detrás de mejoramiento iterativo es desarrollar un sistema de programas de manera incremental, permitiéndole al desarrollador sacar ventaja de lo que se ha aprendido a lo largo del desarrollo anterior, incrementando, versiones entregables del sistema.
Ventajas:

- Provee de soporte para determinar la efectividad de los procesos y de la calidad del producto.
- Permite estudiar y después mejorar y ajustar el proceso para el ambiente en particular.

Desventajas:

- Debido a la interacción con los usuarios finales, cuando sea necesaria la retroalimentación hacia el grupo de desarrollo, utilizar este modelo de desarrollo puede llevar a avances ser extremadamente lento.
- Por la misma razón no es una aplicación ideal para desarrollos en los que de antemano se sabe que serán grandes en el consumo de recursos y largos en el tiempo.
- Al requerir constantemente la ayuda de los usuario finales, se agrega un costo extra a la compañía, pues mientras estos usuarios evalúan el software dejan de ser directamente productivos para la compañía.

Administración de Proyectos

¿Que es un proyecto?

- Están orientados a objetivos
- Tienen una duración limitada en el tiempo, tienen principio y fin.
- Implican tareas interrelacionadas basadas en esfuerzos y recursos.
- Son únicos

Los proyectos tienen una orientación a los objetivos

- Los proyectos están dirigidos a obtener resultados y ello se refleja a través de objetivos.
- Los objetivos guían al proyecto
- Los objetivos deben ser ambiguos
- Un objetivo claro no alcanza…debe ser también alcanzable.

¿Que es la administración de proyectos?

Administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.

Administrar un proyecto incluye:

- Identificar los requerimientos
- Establecer objetivos claros y alcanzables
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders).

Es esencial una buena gestión de proyectos de software para que los proyectos de ingeniería de software se desarrollen en tiempo y según lo presupuestado.

Lista de errores más comunes en la administración de proyectos de software:

- Motivación débil.
- Empleados problemáticos incontrolados.
- Oficinas repletas y ruidosas
- Abandono de la planificación bajo presión.
- Escatimar en las actividades iniciales.
- Escatimar en el control de calidad.
- Cambio en las prestaciones.
- Pérdida de tiempo en el inicio difuso
- Insuficiente input del usuario.
- Planificación agresiva.
- Agregar gente a un proyecto retrazado.

Planificación del proyecto

La gestión efectiva de un proyecto de software depende de planificar completamente el progreso del proyecto. El gestor del proyecto debe anticiparse a los problemas que puedan surgir, así como preparar soluciones a esos problemas. Un plan, preparado al inicio de un proyecto, debe utilizarse como un conductor para el proyecto. Este plan inicial debe ser el mejor posible de acuerdo a la información disponible. Este evolucionara conforme el proyecto progrese y la información sea mejor.

Este plan evolucionara conforme el proyecto progrese y la información sea mejor y mas clara.
El proceso de planificación se inicia con una valoración de las restricciones que afectan al proyecto (fecha de entrega, personal disponible, presupuesto global, etc.).
Se debe llevar a cabo un juntos de estimación de los parámetros del proyecto, entonces se definen los hitos del progreso y productos a entregar. En ese momento el proceso entra en un ciclo. Se prepara un calendario para el proyecto y ls actividades definidas en el calendario se inician o se continúan. Después de algún tiempo, se revisa el plan y se señalan discrepancias.
Cuando se dispone de mas información, los gestores del proyecto revisan las suposiciones del proyecto y la agenda. Si el proyecto se retrasa tienen que renegociar con el cliente las restricciones del mismo y las entregas.
Esta claro que los gestores de proyecto con mas experiencia ya saben que no todo ira bien, durante el proyecto siempre surgen problemas, las suposiciones iniciales y el calendario deben ser mas bien pesimista que optimistas.
Debe haber suficiente holgura para que las contingencias en el plan, las restricciones del proyecto y los hitos no se tengan que negociar cada que se efectúa un ciclo en el plan.

El plan de proyecto
“El plan de proyecto provee una herramienta de comunicación estándar a través del ciclo de vida de un proyecto.”
El plan de proyecto documenta:

- ¿Qué es lo que hacemos?
- ¿Cuándo lo hacemos?
- ¿Cómo lo hacemos?
- ¿Quién lo va a hacer?

El plan de proyecto fija los recursos disponibles, divide el trabajo y crea un calendario de trabajo. En algunas organizaciones el plan de proyecto es un único documento que incluye todos los diferentes tipos de planes (Plan de calidad, plan de validación, plan de gestión de configuración, etc.), y en otros casos este plan solo se refiere al proceso de desarrollo y los otros pueden estar referenciados.
Durante la planificación se debe:

- Armar la infraestructura del proyecto y su entorno
- Asegurar el entrenamiento que haga falta
    - Capacitación General
    - Capacitación basado en roles específicos
    - Capacitación de dominios específicos

Hitos y entregas
Cuando se planifica un proyecto, se debe establecer una serie de hitos (puntos finales de una actividad del proceso del software). En cada uno debe existir una salida formal, como un informe que se debe presentar al gestor. Los informes de hitos no deben ser documentos amplios. **Los hitos deben representar el fin de una etapa lógica en el proyecto.**
Una entrega es el resultado del proyecto que se entrega al cliente. De forma general se entrega al final de una fase principal del proyecto, como la especificación, el diseño, etc. Como regla **general las entregas son hitos, pero los hitos no son necesariamente entregas.**

Work Breakdown Structure (WBS) o Estructura de desglose de tareas (EDS)

- Un arreglo jerárquico de los elementos de una actividad para la cual un conjunto de estimaciones pueden ser determinadas
- Una herramienta para la planificación, la administración y el reporte de un proyecto
- Una herramienta para separar elementos de un proyecto para fines contractuales/financieros o separar áreas de responsabilidad

Usando Breakdown Structures

- El líder de proyecto decide sobre el enfoque de arquitectura y en nivel de detalle del WBS
- Un buen WBS es una excelente herramienta para planificar y generar reportes.

¿Cuándo esta listo el WBS?

- Los elementos del WBS (tareas) son medibles
- Los eventos que marcan el Comienzos / Fin están claramente definidos
- Cada "actividad" tiene un entregable
- Tiempos / costos pueden ser estimados
- Las duración de las Actividades tienen limites aceptables
- Las asignaciones pueden ser hechas independientemente

Gestión de riesgos

Una tarea importante del gestor de proyectos es anticipar los riesgos que podrían afectar a la programación del proyecto o a la calidad del software a desarrollar y emprender acciones para evitar estos riesgos.

Los resultados de este análisis de riesgos se deben documentar a lo largo del plan del proyecto junto con el análisis de consecuencias cuando el riesgo ocurra. Identificar estos y crear planes para minimizar sus efectos en el proyecto se llama **gestión de riesgos.**

De forma simple, se puede definir un riesgo como una probabilidad de que una circunstancia adversa ocurra. Los riesgos son una amenaza para el proyecto, para el software que se esta desarrollando y para la organización.

Los riesgos se pueden clasificar en estas categorías:

- **Riesgos del proyecto**: afectan la calendarización o los recursos del proyecto.
- **Riesgos del producto**: afectan a al calidad o al rendimiento del software que se esta desarrollando.
- **Riesgos del negocio**: afectan a la organización que desarrolla el software.

La gestión de riesgos es muy importante para los proyectos de software ya que tienen requerimientos ambiguamente definidos, es difícil estimar el tiempo y los recursos para el desarrollo del mismo.

El proceso de gestión del riesgo comprende varias etapas:

1- Identificación de los riesgos: Identificar los posibles riesgos.
2- Análisis de riesgos: Valorar la probabilidad y consecuencia de los riesgos.
3- Planificación de riesgos: Crear planes para abordar los riesgos.
4- Supervisión de riesgos: Valorar los riesgos de forma constante y revisar los planes para la mitigación de riesgos tan pronto como la información de los riesgos este disponible.

El proceso de gestion de riesgos es un proceso iterativo.

Estrategias para tratar los riesgos:

- Estrategia de prevención: Siguiendo estas estrategias, la probabilidad de que el riesgo aparezca se reduce.
- Estrategia de minimización: Siguiendo estas estrategias, se reducirá el impacto del riesgo.
- Estrategia de contingencia: Siguiendo estas estrategias, nos permite estar preparados para lo peor y tener una estrategia para cada caso.

Estimaciones

Antes de que el proyecto comience, el gestor del proyecto y el equipo de software deben estimar el trabajo que habrá de realizarse, los recursos que se requerirán y el tiempo que transcurrirá desde el principio hasta el final.
La estimación de recursos, costo y programa de trabajo para una tarea de ingeniería de software requiere experiencia, acceso a buena información histórica (métricas) y el valor para comprometerse con predicciones cuantitativas cuando la información cualitativa es todo lo que existe.
El problema de las estimaciones es que es difícil hacer predicciones hacia el futuro.
¿Que se estima?

- Tamaño
- Esfuerzo
- Calendario
- Costo
- Recursos

La estimación del costo y el esfuerzo nunca será una ciencia exacta. Demasiadas variables (humanas, técnicas, ambientales, políticas, etc.) pueden afectar el costo final del software y el esfuerzo aplicado a desarrollarlo. Sin embargo, la estimación del proyecto de software se puede transformar de una práctica oscura en una serie de pasos sistemáticos que proporcionan estimaciones con riegos aceptables. Para lograr estimaciones confiables de costo y esfuerzo tenemos varias opciones:
1 - Demorar la estimación hasta más tarde en el proyecto.
2 - Basar las estimaciones en proyecto similares que hayan sido completados.
3 – Emplear técnicas de descomposición relativamente simples para generar estimaciones de costo y esfuerzo del proyecto.
4 – Utilizar uno o mas modelos empíricos en la estimación de costo y esfuerzo.

La primera opciones no es practica, las estimaciones se tienen que producir por adelantado. La segunda opción puede funcionar relativamente bien si el proyecto en curso es muy similar a los previos, el problema es que hay mas variables en juego como el equipo, el clientes, etc.
Las opciones restantes son enfoques viables para la estimación de proyectos de software.

Modelos de Estimación

Líneas de código

La métrica de tamaño tradicional para estimar el esfuerzo de desarrollo y productividad ha sido LOC (Lines Of Code) . Se han propuesto varios modelos de estimación, la mayoría de ellos son funciones de las líneas de código o de las miles de líneas de código que tendrá el software a desarrollar.
Los principales problemas de utilizar líneas de código como métrica para estimación del esfuerzo son la falta de una definición universal de línea de código, su dependencia con el lenguaje de desarrollo y la dificultad de estimar en fases tempranas del desarrollo la cantidad de líneas que tendrá una aplicación.

Puntos de función

El análisis por puntos de función es un método para cuantificar el tamaño y la complejidad de un sistema software en términos de las funciones de usuario que este desarrolla (o desarrollará). Esto hace que la medida sea independiente del lenguaje o herramienta utilizada en el desarrollo del proyecto
El enfoque de puntos de función tiene características que permiten superar los principales problemas de utilizar líneas de código como métrica del tamaño del software. Primero, los puntos de función son independientes del lenguaje, herramientas o metodologías utilizadas en la implementación; por ejemplo, no tienen que considerar lenguajes de programación, sistemas de administración de bases de datos, hardware, o cualquier otra tecnología de procesamiento de datos. Segundo, los puntos de función pueden ser estimados a partir de la especificación de requisitos o especificaciones de diseño, haciendo posible de este modo la estimación del esfuerzo de desarrollo en etapas tempranas del mismo.

Métodos utilizados para la estimación

Juicio experto: Puro

- Un experto estudia las especificaciones y hace su estimación.
- Se basa fundamentalmente en los conocimientos del experto.
- Si desaparece el experto, la empresa deja de estimar

Juicio experto: Wideband Delphi

- Un grupo de personas son informadas y tratan de adivinar lo que costará el desarrollo tanto en esfuerzo, como en duración.
- Las estimaciones en grupo suelen ser mejores que las individuales.
- Se dan las especificaciones a un grupo de expertos.
- Se les reúne para que discutan tanto el producto como la estimación.
- Remiten sus estimaciones individuales al coordinador.
- Cada estimador recibe información sobre su estimación, y las ajenas pero de forma anónima.
- Se reúnen de nuevo para discutir las estimaciones.
- Cada uno revisa su propia estimación y la envía al coordinador.
- Se repite el proceso hasta que la estimación converge de forma razonable.

Analogía
Consiste en comparar las especificaciones de un proyecto, con las de otros proyectos. Se compara, usuarios, complejidad, tamaño, tecnología, SO, etc.

Calibraciones y Historical Data

- Las calibraciones son solamente usadas para convertir las "cuentas" a estimados, líneas de código a esfuerzo, use cases a calendario, requerimientos a número de casos de prueba, etc.
- Estimaciones siempre involucra algún tipo de calibración, se implícito o directa.
- Las estimaciones pueden ser calibradas usando cualquiera de estos tres tipos de datos.
    - Datos de la Industria (*Industry data):* datos de otras organizaciones que aportan al mercado y desarrollan productos con algún grado de semejanza y que permite una comparación básica.
    - *Datos Históricos (Historical data):* datos de la organización de proyectos que se desarrollaron y ya cerrados.
    - *Datos del Proyecto (Project data):* datos del proyecto pero de etapas anteriores a la que se esta estimando.

Estimaciones de Tamaño

- El número que más se busca en las estimaciones de software es el tamaño del software a ser construido.
- El tamaño puede se expresado en LOC (Lines Of Code), puntos de función, nro de reqs, nro de web pages, casos de uso u otra medida.
- Hay otros factores: complejidad, madurez, tecnología y GENTE.

Poker estimation
Combina opinión de experto, analogía y desagregación.
1. Defina la lista de actividades, módulos, use cases, etc.
2. Acuerde y consensúe en la más simple . Ej. Tarea Z
3. Asigne tamaño/complejidad 1 a la tarea Z
4. Ejecute un wide band delphi para estimar complejidad/tamaño del resto de la lista, comparado con la más simple y solo asignado valores de COMPLEJIDAD de la serie Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144….). Se puede hacer como una ronda de cartas, si le parece simule una "mano de…."
5. Estime el esfuerzo requerido para llevar a cabo la tarea Z y aplique el multiplicado a todas las actividades

Método basado en los recursos (Parkinson)
En la estimación consiste en ver de cuanto personal y durante cuanto tiempo se dispone de él, haciendo esa estimación.
En la realización: "El trabajo se expande hasta consumir todos los recursos disponibles"

Método basado en el mercado

- Lo importante es conseguir el contrato.
- El precio se fija en función de lo que creemos que esta dispuesto a pagar el cliente.
- Si se usa en conjunción con otros métodos puede ser aceptable, para ajustar la oferta.
- Peligroso si es el único método utilizado

Basado en producto o proceso

- Bottom-up
  Se descompone el proyecto en las unidades lo menores posibles.
  Se estima cada unidad y se calcula el coste total.
- Top-Down
  Se ve todo el proyecto, se descompone en grandes bloques o fases.
  Se estima el coste de cada componente.

La Calendarización de los proyectos de software

La calendarización del proyecto de software, es una actividad que distribuye estimaciones de esfuerzo a través de al duración planificada del proyecto al asignar el esfuerzo a tareas especificas de ingeniería de software. La calendarización, evoluciona a lo largo del tiempo.
Durante las primeras etapas de la planificación del proyecto se desarrolla una calendarización macroscópica. Este tipo de calendarización identifica las principales actividades del marco de trabajo del proceso y las funciones de producto a las que se aplican. Conforme el proyecto transcurre, cada entrada en la calendarización macroscópica se refina en una calendarización detallada. Aquí se identifican y calendarizan tareas especificas del software (requeridas para completar una actividad).

Al igual que otras áreas de ingeniería del software, varios principios básicos guían la calendarización de los proyectos.
Compartimentación: El proyecto debe dividirse en compartimientos en varias actividades, acciones y tareas manejables.
Independencia: Se debe determinar la interdependencia de cada actividad, acción o tarea compartimentada. Algunas tareas deben ocurrir en secuencia mientras que otras pueden ocurrir en paralelo.
Asignación de tiempo: A cada tarea por calendarizar se le debe asignar un cierto número de unidades de trabajo.
Validación del esfuerzo: Todo proyecto tiene un número definido de personas en el equipo de software. Conforme ocurre la asignación de tiempo, el gestor de proyecto debe asegurarse de que en un tiempo dado no se han asignado mas que el numero de personas calendarizadas.
Definición de responsabilidades: Toda tarea calendarizada se le debe asignar a un miembro específico del equipo.
Definición de resultados: Toda tarea calendarizada debe tener un resultado definido.
Definición de hitos: Cualquier tarea o grupo de tareas debe estas asociado con un hito del proyecto.

Monitoreo y Control

En administración de proyectos el término “control” se refiere a comparar el progreso con respecto al plan, así acciones correctivas pueden ser tomadas cuando desviaciones significativas ocurren con respecto al plan.

Información y datos son el componente primario del control

Para saber que el proyecto esta bajo control se debe evaluar:

- Se están alcanzando los hitos del proyecto:
  - A tiempo
  - Con los recursos estimados
  - Con un nivel de calidad
  - Continúa siendo aceptable económicamente

En el momento que se observan desviaciones hay que replanificar/renegociar el plan de proyecto.

Actividades de Seguimiento y control

- Reuniones semanales de seguimiento
- Encuentros con el cliente
- Revisiones mensuales del proyecto
- Check-list de fin de fase
- Revisiones del plan
- Auditorías
- Reuniones de Postmortem
-

¿Por que hacer Tracking?

- Para minimizar riesgos
- Para conocer y aceptar la realidad
- Para tomar acciones correctivas
- Para evitar ser “bomberos”
- Para mejorar ciertas áreas, ej.
  - Revisión entre pares
  - Errores en el código
  - Calidad del producto
  -

Encuentros con el cliente

- Acordar la frecuencia y agendar los encuentros
- Reportes de progreso, identificar problemas potenciales, monitorear las acciones
- Identificar y administrar
  - Cambios en los requerimientos
  - Cambios al Schedule
- Hacer una minuta de la reunión (e-mail)
- Los clientes pueden requerir reportes regulares en un formato especifico.

Auditorias

- Alcance de la auditoria
  - Minutas de reunión
  - Métricas
  - Seguimiento de Acciones
  - Control de cambios

    - Adm. de configuraciones (Físicas y Funcionales)
- Salida de la auditoria
    - Reporte de la auditoria
    - Acciones Correctivas

Unidad 2: Métricas de Software

La medición permite obtener una visión del proceso y el proyecto pues proporciona un mecanismo para lograr una evaluación objetiva.

¿Qué son las Métricas de Software?
"La aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo de software y sus productos para suministrar información relevante a tiempo, así el líder de proyecto junto con el empleo de estás técnicas mejorará el proceso y sus productos"

Medida: proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto.

Medición: es el acto de determinar una medida.

Métrica: una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.

Indicador: es una métrica o una combinación de métricas que proporcionan una visión profunda del proceso del software, del proyecto de software, o del producto en sí.

¿Por qué medimos?
Para ganar control sobre nuestros proyectos, y en base a esto lograr supervisarlo: s**eñalar**
Para poder estimar nuevos proyectos: **Predecir**
Para tener noción sobre los costos: **Evaluar**
Para poder **mejorar**

Tipos de Métricas
Métricas directas o primitivas: aquellas que para medir un atributo de una entidad sólo precisan de dicho atributo.
Métricas indirectas: aquellas que no miden un atributo directamente, sino que para medir un atributo de una entidad precisan medir más de un atributo de dicha entidad, es decir, combinan varias métricas directas.

El dominio de las métricas del software se divide en:

- Métricas de proceso.
- Métricas de proyecto.
- Métricas de producto.

¿Por qué es necesario medir el proceso?

- Conocer y mejorar sus costos, disminuir el tiempo de construcción (tiempo de mercado), mientras se entregan productos con una calidad definida y las demandas de mantenimiento son apropiadas.
- Mejorar continuamente la performance. Para reducir los riesgos se necesita conocer las habilidades de los miembros del equipo de desarrollo y la experiencia del staff en general, y asegurarse que se cuenta con los recursos para competir en un determinado dominio.
- Debería estar disponible para introducir nuevas tecnologías y poder determinar su costo-beneficio.

Las métricas del proceso se recopilan en el curso de todos los proyectos y durante largos periodos. Su objetivo es proporcionar un conjunto de indicadores de proceso que conduzcan a la mejora de los procesos de software de largo plazo.
Una forma de mejorar cualquier proceso es medir sus atributos específicos, desarrollar un conjunto de métricas significativas con base en dichos atributos y luego emplear las métricas para ofrecer indicadores que conducirán a una estrategia de mejora.
Algunos ejemplos de métricas de proceso son:

- Errores descubiertos antes de liberar el software.
- Los defectos que detectan y reportan los usuarios
- El esfuerzo humano gastado.
- El tiempo de la planificación consumido.
- Reuso de datos.
- Propagación de errores de fase a fase.

¿Por qué es necesario medir el proyecto?

Porque un proyecto debería ser entregado con:

- Las capacidades funcionales y no funcionales requeridas por el cliente/usuario.
- Las restricciones impuestas.
- El presupuesto y tiempo planificado
- Un producto que cumpla ciertas características de despliegue, operacionales y de mantenimiento.

Las métricas de proyecto, permiten que un gestor de proyecto de software valore el estado de un proyecto en curso, rastree los riesgos potenciales, descubra las áreas problemas antes de que se vuelvan criticas, ajuste el flujo de trabajo o las tareas, evalúe la habilidad del equipo del proyecto para controlar la calidad del los productos de trabajo de software.
La primera aplicación de las métricas del proyecto en la mayoría de los proyectos de software ocurre durante la estimación. Las métricas recopiladas de los proyectos previos se aprovechan como base desde la cual se realizan estimaciones de esfuerzo y tiempo para el trabajo actual.
La finalidad de las métricas del proyecto es doble, primero se emplean para minimizar el tiempo de desarrollo haciendo los ajustes necesarios para evitar demoras y reducir los problemas y riesgos potenciales. Segundo, se utilizan para valorar la calidad del producto sobre una base actual y cuando es necesario, modificar el enfoque técnico para mejorar la calidad.

Ejemplo de métricas del proyecto:

- Esfuerzo/Tiempo por Tarea de Ing. Sw
- Errores no cubiertos por hora de revisión
- Fechas de entrega reales vs programadas
- Cambios (número) y sus características
- Distribución del esfuerzo sobre tareas de Ing. Sw

¿Por qué es necesario "Medir los Productos"?
Durante el proceso de desarrollo se generan un conjunto de artefactos (componentes y modelos) que deberían ser controlados para garantizar:

- Que cumplen con los requerimientos del cliente
- Que están libres de error
- Que cumplen con los criterios de calidad existentes.
- Que se realizaron bajo los procedimientos de calidad

Métricas básicas

- Esfuerzo
- Calendario
- Defectos
- Tamaño (funcionalidad, LOC, paginas, etc)

<u>Mediciones del Software</u>
La medición de software se clasifica en dos categorías.

1- Medidas indirectas del proceso de software (ej. Costo y esfuerzo aplicados) y del producto (ej. LDC producidas, rapidez de ejecución, defectos reportados)
2- Medidas indirectas del producto que incluyen funcionalidad, calidad, complejidad, eficiencia, confiabilidad, falibilidad de mantenimiento, etc.

Las métricas del proyecto se consolidan con el fin de crear métricas del proceso que sean públicas para la organización de software como un todo.
Si las mediciones se normalizan, es posible crear métricas de software que posibiliten la comparación a promedios organizacionales más amplios. De esta forma, las métricas orientadas tanto al tamaño como a la función están normalizadas.

Principio de medición
Formulación: La derivación de medidas y métricas apropiadas para la representación del software que se considera.
Recolección: El mecanismo con que se acumulan los datos necesarios para derivar las métricas formuladas.
Análisis: El calculo de las métricas y la aplicación de herramientas matemáticas.
Interpretación: La evaluación de las métricas en un esfuerzo por conocer mejor la calidad de representación.
Retroalimentación: Recomendaciones derivadas de la interpretación de las métricas del producto transmitidas al equipo del software.

Métricas orientadas al tamaño

Las métricas orientadas al tamaño no se aceptan universalmente como la mejor forma de medir un proceso de software. La mayor parte de la controversia gira en torno al uso de líneas de código como medida clavé.
Muchos dicen que son algo fácil de contar y que esta presente en todos los proyectos, mientas que por el otro lado muchos dicen que no son algo estable, ya que la cantidad de líneas de código dependen mucho del lenguaje de programación utilizado.
Algunas metricas de tamaño son:

- Errores por KLDC.
- Defectos por KLDC
- Costo por KLDC.
- Paginas documentadas por KLDC.
- KLDC por persona-mes

Métricas orientadas a la función

Las métricas de software orientadas a la función emplean como un valor de normalización una medida de la funcionalidad que entrega la aplicación. La métrica orientada a la función utilizada con mayor amplitud es el punto de función.
El cálculo del punto de función se basa en características del dominio de información y la complejidad del software.
El PF al igual que LDC es controversial. Los partidarios afirman que el PF es independiente del lenguaje de programación, y que se basa en datos que es más probable que se conozcan temprano en la evolución de un proyecto, lo que hace al PF más atractivo como enfoque de estimación.
Los detractores afirman que el método requiere cierta prestidigitación en cuanto a que el cálculo se basa en datos subjetivos más que objetivos y que el PF no tiene significado físico directo, es solo un número.

Métricas de Calidad

La calidad de un sistema, aplicación o producto es tan bueno como:

- los requisitos que describen el problema,
- el diseño que modela la solución,
- el código que conduce a un programa ejecutable,
- las pruebas que ejercitan el software para detectar errores.

Aunque existen muchas medidas de la calidad del software, la corrección, la facilidad de mantenimiento, la integridad y la facilidad de uso ofrecen indicadores útiles para el quipo del proyecto.

Corrección: Un programa debe operar correctamente o proporcionara poco valor para sus usuarios. La corrección es el grado en que el software desempeña la función para la que fue creado. La medida más común para la corrección es defectos por KLDC.

Facilidad de Mantenimiento El mantenimiento del software justifica más esfuerzos que cualquier otra actividad de la ingeniería del software. La facilidad de mantenimiento es la sencillez con la que un programa puede corregirse si se encuentra un error, adaptarse si su entorno cambia, o mejorar si el cliente desea un cambio en los requerimientos.
No existe una forma directa de medir la facilidad de mantenimiento, por lo que se deben usar métricas indirectas. Ej. Tiempo medio de cambio.

Integridad: La integridad del software se ha vuelto cada vez más importante en la edad de los hackers. Este atributo mide la habilidad de un sistema para resistir ataques.
La medición de la integridad necesita definir 2 atributos más: seguridad y amenaza. Amenaza es la probabilidad que un ataque ocurra durante un tiempo determinado y seguridad es la probabilidad de que se repela el ataque de un tiempo específico.

Facilidad de Uso: Con frecuencia un programa que no es fácil de usar esta condenado al fracaso, incluso si las funciones que realiza son valiosas. La facilidad de uso es un intento por cuantificar la sencillez de la aplicación al utilizarla.

Integración de métricas
Si no se mide, no existe una forma real de determinar si se esta mejorando, y si no se mejora si esta perdido.
Si el proceso con el cual se desarrolla puede mejorarse, se producirá un impacto directo en lo sustancia. Pero para establecer objetivos de mejora, es preciso comprender el estado actual del desarrollo de software. Por lo tanto, la medición se emplea para establecer una línea base del proceso a partir de al cual se evalúan la mejoras.

Establecimiento de una línea base
Con el establecimiento de una línea base de métricas e obtienen beneficios en los ámbitos del proceso, del proyecto y del producto. Incluso la información que se recopila no necesita ser fundamentalmente diferente. Las mismas métricas pueden servir a muchos maestros. La línea base de métricas consiste de datos recopilados en proyectos previos de desarrollo de software y pueden ser tan simples como una tabla o tan complejos como una base de datos.
Para ser un auxiliar eficaz en el proceso de mejora o de costo y esfuerzo los datos de la línea base deben tener los siguientes atributos:

- Los datos deben ser razonablemente precisos.
- Los datos deben recopilarse para tantos proyectos como sea posible.
- Las medidas deben ser consistentes.
- Las aplicaciones deben ser similares al trabajo que se estimara.

Unidad 3: Gestión de Configuración de Software

Conceptos Introductorios a la Gestión de Configuración de Software
El cambio es inevitable cuando se construye software, y el cambio aumenta el grado de confusión ente los ingenieros de software que trabajan en un proyecto. La confusión surge cuando los cambios no se analizan antes de realizarlos, no se registran antes de implementarlos, no se reportan a quines deben saberlo o no se controlan en una forma que mejorará la calidad y reducirá el error.
El arte de coordinar el desarrollo de software para minimizar la confusión se llama gestión de la configuración. La gestión de la configuración es el arte de identificar organizar y controlar modificaciones al software que se construye por medio de un equipo de programación. La meta es maximizar la productividad al minimizar las equivocaciones.
La GCS, es una actividad protectora que se aplica a lo largo del proceso del software. Como las actividades de GCS pueden ocurrir en cualquier momento, se desarrollan para:

- Identificar el cambio

- Controlar el cambio
- Garantizar que el cambio se implementara bien
- Reportar los cambios a otros que estén interesados

El propósito de la GCS es establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo

Las ventajas de realizar la gestión de configuración de software son:

- Aumenta la protección contra cambios innecesarios
- Mejora de la visibilidad del estado del proyecto y sus componentes
- Aumenta la auto responsabilidad
- Disminuye los costos por retrabados
- Disminuye el tiempo de desarrollo
- Aumenta la calidad

Los cambios pueden surgir por diversas causas, las cuales generalmente se encuadran entre las siguientes:

- Nuevas condiciones en el negocio o mercado dictan cambios en los req's
- Nuevas necesidades del cliente demandan la modificación de los datos que produce el sistema
- La reorganización o el crecimiento o reducción del negocio provocan cambios en las prioridades del proyecto.
- Restricciones presupuestales o de calendarización

¿Como se hace la GCS?
Como son muchos los productos de trabajo que se producen cuando se construye software, cada uno debe identificarse de forma individual. Una vez hecho esto se establecen los mecanismos de control de versión y cambio. El proceso se audita para garantizar que la calidad se conserva conforme el cambio se realiza y que quienes tienen necesidad de conocerlo reciben información acerca de los cambios mediante los informes respectivos.

Problemas que soluciona la GCS

- Pérdida de un componente
- Pérdida de cambios (el componente que tengo no es el último)
- Doble mantenimiento
- Superposición de cambios
- Cambios no validados

<u>Versiones, variantes y release</u>
Una versión es una instancia de un elemento de Configuración. El término se usa para señalar a un elemento de Configuración del software que tiene un conjunto definido de características funcionales.
Se define variante como una versión que es una alternativa a otra versión. Las variantes pueden permitir a un elemento de Configuración satisfacer requerimientos en conflicto. Una variante es una nueva versión de un elemento que será añadida a la Configuración sin reemplazar a la versión anterior.
Release es una versión que se libera y se entrega al usuario o cliente.

Actividades Relacionadas con la Gestión de Configuración

1 - Implementación del Proceso: Se desarrolla un Plan de Gestión de Configuración que describe las actividades de Gestión de Configuración, los procedimientos y el cronograma para su realización, y los responsables de dichas actividades. Dicho plan debe ser documentado e implementado.

2 - Identificación de la Configuración: Se establece un esquema de identificación de los elementos de software y sus versiones a ser controlados por el proyecto.

3 - Control de la Configuración: Se identifican y registran las solicitudes de cambio, se analiza y evalúa los cambios, se aprueba o rechaza la solicitud, se implementa, verifica y distribuye el elemento de software modificado.

4 - Contabilidad de Estado de la Configuración: Se preparan registros de Gestión y reportes de estado que muestren el estado e historia de los elementos de software controlados, incluyendo líneas base.

5 - Evaluación de la Configuración: Se determina y asegura que los elementos de software sean funcionalmente (versus sus requerimientos) y físicamente completos (es decir, si su diseño y Código reflejan una descripción técnica actualizada).

6 - Gestión de actualización y distribución: Se controla formalmente la actualización y distribución de los productos de software.

El rol de las baselines y su administración

Una línea base, es una especificación o producto que se ha revisado formalmente y se esta de acuerdo con los resultados y que a partir de ahí sirve como la base para el desarrollo ulterior y que puede cambiarse solo por medio de procedimientos formales de control del cambio.
Antes de que un elemento de configuración del software se convierta en línea base, es posible realizar el cambio rápida e informalmente. Sin embargo una vez establecida una línea base, los cambios se pueden realizar pero se debe aplicar un procedimiento especifico formal para evaluar y verificar cada uno.
En la ingeniería de software, una línea base es un hito en el desarrollo del software. Se marca una línea base para la entrega de uno o más elementos de configuración del software que se han aprobado como consecuencia de una revisión técnica formal.
Las líneas base pueden ser:

- De especificación (reqs, diseño)
- De producto que han pasado por un control de calidad definido previamente

Se puede decir que una línea base es un conjunto de ítems de configuración de software revisados, corregidos y aprobados.

Elementos de configuración del Software

Existen 4 elementos que deben estar presentes cuando se desarrolla un sistema de gestión de la configuración:

- Elementos de componentes: conjunto de herramientas acopladas dentro de un sistema de gestión de archivos (EJ. Una base de datos) que permiten el acceso a la gestión de cada elemento de configuración de software.

- Elementos de procesos: Serie de procedimientos y tareas que definen un enfoque eficaz con el cual gestionar el cambio para todos los participantes en la gestión, ingeniería y utilización del software de computadora.
- Elementos de construcción: Conjuntos de herramientas que automatizan la construcción del software al asegurar que se ha ensamblado un conjunto adecuado de componentes validos (es decir una versión correcta)
- Elementos humanos: La implementación de una GCS eficaz requiere que el equipo de software utilice un conjunto de herramientas y características de procesos.

El deposito de elementos de configuración de software
El depósito, es una base de datos que actúa como centro tanto de la acumulación como del almacenamiento de la información de ingeniería del software. El papel de los ingenieros de software, es interactuar con el depósito mediante las herramientas que tiene integradas.
El deposito ayuda a un equipo de software a manejar el cambio en una forma eficaz.
El deposito ayuda con las siguientes actividades:

- *Integridad de los datos*: Incluye funciones para validar las entradas al deposito, garantizar la consistencia entre los objetos y automatizar las modificaciones en cascada.
- *Compartir Información*: Ofrece un mecanismo para distribuir la información entre múltiples desarrolladores y herramientas, manejar y controlar los accesos a los datos por parte de múltiples usuarios y cerrar y abrir los objetos de modo que los cambios no sean trasladados inadvertidamente hacia otros.
- *Integración de herramientas*: Establece un modelo de datos al que se puede tener acceso mediante muchas herramientas, controlar el acceso a los datos y realizar funciones de gestión de configuración.
- *Estandarización de los documentos*: Es la definición de los objetos en la base de datos que conduce directamente a un enfoque estándar para la creación de documentos de la ingeniera de software.

<u>Identificar los objetos en la configuración del software</u>
El control y la gestión de elementos de configuración de software requiere nombrar cada uno por separado y luego organizarlo mediante un enfoque orientado a objetos. Es posible identificar dos tipos de objetos: *básicos y agregados.*
Un objeto básico es una unidad de información creada por un ingeniero de software durante el análisis, diseño, código o las pruebas.
Un objeto agregado es una colección de objetos básicos y otros objetos agregados.

<u>Control de Versiones</u>
El control de versiones combina procedimientos y herramientas para gestionar diferentes versiones de objetos de configuración que se crean durante el proceso del software. Un sistema de control de versiones implementa cuatro grandes capacidades:

- Una base de datos del proyecto (deposito) que guarda todos los objetos de configuración relevantes.
- Una capacidad de gestión de versione que almacena todas las versiones de un objeto de configuración.
- Una facilidad de hechura que permita al ingeniero de software recopilar todos los objetos de configuración relevantes y construir una versión especifica del software.

Además, por lo general los sistemas de control de versión y de cambio implementan una capacidad de seguimiento de complicaos que permite al equipo registrar y hacer seguimiento del estado de todos los conflictos destacados que se asocian con cada objeto de configuración.

El versionado nos permite:

- Fácil acceso a todos los componentes
- Reconstrucción de cualquier versión
- Registro de Historia
- No se pierden
- Centralización (no se requiere la presencia del dueño)
- Información de resumen

Gestión de Cambios

En un proyecto, el cambio incontrolado conduce rápidamente al caos. El control de cambios combina procedimientos humanos y herramientas automatizadas.
El proceso de control de cambio es el siguiente: Se emite una solicitud de cambios y se estima para evaluar los meritos técnicos, potenciales efectos colaterales, el impacto global sobre otros objetos de configuración y funciones del sistema, y el costo del cambio. Los resultados de la evaluación se presentan como un informe de cambio, que lo utiliza una autoridad del control del cambio, una persona o un grupo de personas toman la decisión final acerca del estado y la prioridad del cambio. Se genera una orden de cambio para cada cambio aprobado.
La orden de cambio describe el cambio que se debe realizar, las restricciones insoslayables y los criterios de revisión y auditoria.
Un sistema de control de versiones actualiza el archivo original una vez realizado el cambio. Se utiliza el control de versiones para tener control de acceso y sincronización.
Los controles de cambios formales solo se aplican cuando un elemento ya se convirtió en línea base, antes de esto se realizan actividades de cambio informal.

Auditoria de la configuración

¿Cómo se puede garantizar que el cambio se ha implementado con propiedad?
La respuesta es doble:

- Revisiones técnicas formales
- Auditorias de Configuración

La revisión técnica formal se enfoca en la corrección técnica del objeto de configuración que se ha modificado. Los revisores evalúan el elemento de configuración para determinar su consistencia con otros elementos. Se debe realizar una revisión técnica en casi la mayoría de los cambios triviales.
Una auditoria de configuración, complementa la revisión técnica formal al abordar las siguientes preguntas:

- ¿Se ha realizado el cambio especificado en al orden de cambio? ¿Se agrego algo más?
- ¿Se ha realizado una revisión técnica formal?
- ¿Se ha seguido el proceso d software? ¿Se han aplicado los estándares?
- ¿Se especifico la fecha y el auto del cambio?
- ¿Los demás elementos se han actualizado de forma correcta?

Es decir la auditoria de configuración cumple las siguientes funciones:

- Determinar la semejanza entre el estado actual del sistema y el establecido como línea base.
- Provee el mecanismo para establecer una línea base.

Auditorias físicas de configuración: asegura que lo que está indicado para cada SCI en la línea base o en la actualización se ha alcanzado realmente. Evaluación independiente de los SCI para verificar que el software y su documentación son internamente consistentes y están listos para ser entregados al cliente.
Auditoria funcional de configuración: evaluación independiente de los productos de software, verificando que la funcionalidad y performance reales de cada ítem de Configuración sean consistentes con la especificación de requerimientos de software.

Hay que recordar que:

- Las auditorias cuestan tiempo y dinero
- Deben realizarse desde la primeras etapas de desarrollo.
- Su postergación a etapas posteriores puede llegar a hacer fracasar el proyecto
- Suministra visibilidad y rastreabilidad del ciclo de vida del producto de software

<u>Continuous Integración</u>

- Continuous Integration es una practica de desarrollo donde los miembros de un equipo integran su trabajo frecuentemente, generalmente una persona integra por lo menos diariamente.
- Cada integración es verificada por una herramienta de build automático (incluido un test) para detectar errores de integración tan pronto como sea posible

¿Qué es un build exitoso?

- Tener un build exitoso no es compilar todos los días!!!!
- La definición de “build exitoso” es un tanto agresiva.
    - Los fuentes deben estar en el repositorio (check operation)
    - Cada fuente se compila.
    - Los resultados de los object files son linkeados y se los deja listo para ejecución (jars)
    - Se arranca el sistema y se corre una suite de test cases.
    - Si todo esto se ejecuta sin errores y sin intervención humana, entonces hay un “build exitoso”
- Mientras más exhaustiva sea la suite de test cases más valor genera la integración continua.

Unidad 4: Aseguramiento de la calidad del proceso y del producto

Conceptos generales sobre calidad

Decimos que calidad son todos los aspectos y características de un producto o servicio que se relacionan con su habilidad de alcanzar las necesidades manifiestas o implícitas.

El control de la variación es el corazón del control de calidad. Un fabricante quiere minimizar la variación entre los productos que se producen.

Cuando buscamos la calidad en el software, tratamos de responder estas preguntas:

- ¿Hace lo que el usuario necesita?
- ¿Es lo que el usuario quiere?
- ¿Le soluciona el problema?
- ¿Lo hace como el quiere?
- ¿Se puede construir?
- ¿Es fácil de modificar, de corregir, de extender?
- ¿Se lo puede hacer en tiempo y forma, con costos bajos?
- ¿Me gusta? ¿Le gusta al usuario?

Un Software de calidad satisface:

- Las expectativas del Cliente
- Las expectativas del Usuario
- Las necesidades de la gerencia
- Las necesidades del equipo de desarrollo y mantenimiento

La calidad del software se estructura en tres actividades:

1 – *Garantía de la calidad*: El establecimiento de un marco de trabajo de procedimientos y estándares que conduce a software de calidad.

2 – *Planificación de la calidad*: La selección de procedimientos y estándares adecuados a partir de este marco de trabajo y la adaptación de estos para un proyecto software especifico.

3 – *Control de la calidad*: La definición y fomento de los procesos que garanticen que los procedimientos y estándares para la calidad del proyecto son seguidos por el equipo de desarrollo de software.

¿Por qué ocuparse de la calidad?

- Es un aspecto competitivo
- Es esencial para sobrevivir
- Es indispensable para el mercado internacional
- Equilibrio costo-efectividad
- Retiene clientes e incrementa beneficios
- Es el sello de clase en el mundo de los negocios

El equipo de calidad no esta asociado a ningún grupo de desarrollo, sino que tiene la responsabilidad de la gestión de la calidad de toda la organización. La razón de esto es que los gestores del proyecto deben mantener el presupuesto y la agenda. Si aparecen

problemas, estos pueden verse tentados de comprometer la calidad del producto para mantener su agenda. Un equipo independiente de calidad garantiza que los objetivos organizacionales y la calidad no sean comprometidos por consideraciones de presupuesto o agenda.

Tendencia de la Calidad

La calidad del software se ha mejorado significativamente en los quince últimos años. Una de las razones ha sido que las compañías han adoptado nuevas técnicas y tecnologías como el uso de desarrollo orientado a objetos y el soporte asociado de las herramientas CASE. No obstante también ha habido una mayor conciencia de la importancia de la gestión de la calidad y de la adopción de técnicas de gestión de la calidad para desarrollo en la industria del software.
Sin embargo, la calidad del software es un concepto complejo que no es directamente comparable con la calidad de la manufactura de productos.
Para tener calidad en el software existen algunos problemas:

- Las especificaciones se orientan hacia las características del producto que el consumidor quiere. Sin embargo, la organización desarrolladora también tiene requerimientos (como los de mantenimiento) que no se incluyen en la especificación.
- No se sabe como especificar ciertas características de la calidad (Ej. Mantenimiento, seguridad, eficiencia) de forma no ambigua.
- Es muy difícil redactar especificaciones concretas de software. Por lo tanto, aunque un producto se ajuste a su especificación, los usuarios no lo consideran un producto de alta calidad debido a que no responde a sus expectativas.

Se deben reconocer estos problemas con la especificación del software y se tienen que diseñar procedimientos de calidad que no se basen en una especificación perfecta.

Calidad de proceso y producto

Una suposición subyacente de la gestión de calidad es que la calidad del proceso de desarrollo afecta directamente a la calidad de los productos. Esta suposición viene de los sistemas manufactureros donde la calidad del producto esta íntimamente al proceso de producción.
Sin embargo, el software no se manufactura, sino que se diseña. El desarrollo del software es un proceso más creativo que mecánico donde la experiencia y las habilidades individuales son importantes. La calidad del producto también se ve afectada por factores externos, como la novedad de una aplicación o la presión comercial de sacar un producto rápidamente.
En el desarrollo de software, la relación entre la calidad del proceso y la calidad del producto es muy compleja. Es difícil medir los atributos de la calidad del software, como la mantenibilidad, incluso después de utilizar el software durante un largo periodo. En consecuencia es difícil explicar como influyen las características del proceso en estos atributos.
A pesar de ello, la experiencia nos muestra que la calidad del proceso tiene una influencia significativa en la calidad del software. La gestión de la calidad del proceso debe minimizar los defectos en el software entregado.
La gestión de calidad del proceso implica:

- Definir estándares de proceso, como las revisiones a realizar, cuando llevarlas a cabo, etc.
- Supervisar el producto de desarrollo para asegurar que se sigan los estándares.

- Hacer informes del proceso para el gestor del proyecto y para el comprador del software.

Garantía de la calidad y estándares

La garantía de la calidad es el proceso que define como lograr la calidad del software y como la organización de desarrollo conoce el nivel de calidad requerido en el software. El proceso de aseguramiento de calidad se ocupa ante todo de definir o seleccionar los estándares que deben ser aplicados al proceso de desarrollo software o al producto software.

Podemos definir dos tipos de estándares como parte del proceso de garantía de calidad:

- *Estándares de Producto*: Estos estándares se aplican sobre el producto software que se comienza a desarrollar. Incluyen estándares de documentación, como cabecera de comentarios estándar para la definición de clases y estándares de codificación, que definen como debe utilizarse el lenguaje de programación.
- *Estándares de Procesos*: Estos estándares definen os procesos que deben seguirse durante el desarrollo del software. Pueden incluir definiciones de procesos de especificación, diseño y validación. Así como una descripción de los documentos que deben escribirse en el curso de estos procesos.

Existe una relación muy cercana entre los estándares de producto y los estándares de proceso. Los estándares de producto se aplican a las salidas del proceso software y, en muchos casos, los estándares de proceso incluyen actividades de proceso específicas que garantizan que se sigan los estándares de producto.

Actividades relacionadas con el aseguramiento de la calidad del software

La garantía de la calidad de software se compone de una variedad de tareas asociadas con dos integrantes diferentes: los ingenieros de software que realizan el trabajo técnico y un grupo de aseguramiento de la calidad que tiene la responsabilidad de planificar, supervisar, guardar registros, analizar y reportar la calidad.

Los ingenieros de software abordan la calidad al aplicar sólidos métodos y medidas técnicas, llevar a cabo revisiones técnicas formales y desarrollar pruebas de software bien planificadas.

La misión del grupo de aseguramiento de la calidad es auxiliar al equipo de software a conseguir un producto final con alta calidad.

Las actividades del aseguramiento de la calidad son:

- *Preparar un plan de SQA para un proyecto*: El plan se desarrolla durante la planificación del proyecto y lo revisan todos los participantes. Las actividades de garantía de la calidad del equipo de ingeniería del software y del grupo de SQA las rige el plan. Este identifica las evaluaciones que se realizaran, las auditorias y revisiones para llevar a cabo, los estándares aplicables al proyecto, los procedimientos para el informe y seguimiento de errores, los documentos que debe producir el grupo de SQA y la cantidad de retroalimentación proporcionada al equipo del proyecto software.
- *Participar en el desarrollo de la descripción del proceso de software del proyecto*: El equipo de software selecciona un proceso para el trabajo que habrá de realizarse. El grupo de SQA revisa la descripción del proceso para que concuerde con las políticas organizacionales, los estándares internos de software, los estándares impuestos de manera externa y otras partes del plan de proyecto del software.

- *Revisar las actividades de ingeniera del software para verificar que se ajuste al proceso de software definido*: El grupo de SQA identifica, documenta, sigue las desviaciones del proceso y verifica que se hayan hecho las correcciones.
- *Audita productos de trabajo de software seleccionados para verificar que se ajusten con los definidos como parte del proceso del software*: El grupo de SQA revisa los productos de trabajo seleccionados, identifica, documenta y sigue las desviaciones, verifica que se hayan hecho las correcciones y periódicamente informa de los resultados de su trabajo al gestor del proyecto.

Costos de la Calidad
Incluye los costos asociados en la búsqueda y obtención de la calidad, pueden dividirse en costos asociados con:

- La prevención
- La evaluación
- Los fallos.(solo presentes si existen defectos)

El costo de a calidad incluye todos los costos que genera la búsqueda de la calidad o que demanda el desarrollo de las actividades relacionadas con la calidad.

Modelo de Mejora
En la actualidad, al estar muy ligadas la calidad del proceso con la calidad del producto, muchas empresas de ingeniería de software, han tomado el camino de la mejora de los procesos del software para mejorar su software. La mejora de procesos significa entender los procesos existentes y cambiarlos para mejorar la calidad del producto y/u reducir los costos y tiempo de desarrollo.
No es posible hacer mejoras de procesos que optimicen todos los atributos del proceso de forma simultánea, por ejemplo si se requiere que un proceso de desarrollo rápido entonces es necesario reducir la visibilidad del proyecto.
La mejora de procesos es una actividad cíclica y tiene 3 estados principales:
1 – Proceso de medición de los atributos del proyecto actual o del producto. El objetivo es mejorar las mediciones de acuerdo con las metas de la organización involucrada en el proceso de mejora.
2 – Proceso de análisis. El proceso actual es valorado y se identifican puntos flacos y cuellos de botella. En esta etapa se suelen desarrollar los procesos que describen los modelos de proceso.
3 – Introducción de los cambios del proceso identificados en el análisis.

Muchas veces los cambios en los procesos deben introducirse por otros motivos que no son necesariamente la búsqueda de la mejora.
Hay veces que las organizaciones se enfrentan frente a otros problemas que obligan a modificar el proceso de desarrollo, algunos ejemplos son:

- Muchas personas claves enferman al mismo tiempo.
- Una avería en la computadora de seguridad que deja todas las comunicaciones fuera de servicio durante varias horas.
- Se lleva a cabo una reorganización de la organización, lo que significa que los gestores tienen que invertir gran parte de su tiempo trabajando en cuestiones organizacionales en lugar de la gestión del proyecto.

Aspectos culturales de la mejora de procesos:

- Compromiso en todos los niveles.

- Actitudes y valores del personal pueden necesitar cambios:
  - Capacitación
  - Mecanismos de reconocimiento
  - Estructura Organizacional
  - Comunicación Organizacional
  - Trabajo colaborativo.
- Objetivos medibles y consensuados.
- Confidencialidad y Confianza.

Pasos para la Mejora del Desarrollo de Software en las Organizaciones

- Comprender el estado actual del proceso de desarrollo.
- Desarrollar una visión del proceso deseado.
- Establecer una lista de acciones de mejora del proceso en orden de prioridad.
- Crear un plan para alcanzar las acciones requeridas.
- Comprometer los recursos para ejecutar el plan.
- Comenzar nuevamente en el punto 1.

Algunos modelos para la Mejora de Procesos son:

- **SPICE**: **S**oftware **P**rocess **I**mprovement **C**apability **E**valuation.
- **IDEAL**: **I**nitiating, **D**iagnosing, **E**stablishing, **A**cting, **L**everaging.

Beneficios de la Mejora de Procesos

- Productividad:
  - Se descubren defectos en forma temprana lo que permite reducir substancialmente el re-trabajo.
  - Se reducen los costos de desarrollo y mantenimiento de Software.
  - Se reducen los ciclos de desarrollo lo que produce un incremento en la productividad.
  - Se provee un marco para mediciones y seguimiento
- Calidad:
  - Se reduce la cantidad de defectos informados por los clientes, lo que permite mejorar la Satisfacción del Cliente.
  - Se estima que el 50% de todos los defectos que existen son solucionados cuando comienza una fase con las inspecciones realizadas durante esa fase.
  - Se mejora la relación con los clientes internos a partir de una correcta definición y seguimiento de los requerimientos.
- Planificación:
  - Se reducen los ciclos de Testa la mitad, los ciclos de diseño tienden a ser más largos, pero los tiempos para el código y prueba son más cortos.
  - Se observa una leve mejora del índice de performance (diferencia entre lo planificado y el tiempo real) durante el primer año de aplicación de SPI (Software Process Improvement), sin embargo éste tenderá a mejorar sustancialmente en cada año subsiguiente.
- Cambio Organizacional:
  - Se clarifican roles y responsabilidades.
  - Se disciplina el trabajo.

- Se provee de un marco que mejora la comunicación
- Se produce un cambio organizacional y cultural.
- Se adoptan Técnicas y Métodos.

Procesos de Desarrollo

¿Que son los procesos de desarrollo?

- Conjunto estructurado de actividades para desarrollar un sistema de software
- Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse.
- Debe ser explícitamente modelado si va a ser administrado.

Proceso de Software: Un conjunto de actividades, métodos, prácticas, y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados

¿Qué ocurre cuando el proceso no está definido?

- Las actividades del proceso no están definidas:
  - por lo tanto no se pueden planificar
  - por lo tanto no se pueden controlar
  - por lo tanto no se puede asegurar los resultados

¿Qué se espera de un Proceso de Desarrollo?

- Que sea capaz de evolucionar
- Durante su evolución se limite a las realidades que imponen :
  - La Tecnología
  - Las Herramientas
  - La gente
  - Los patrones organizacionales

Para definirlo se debe responder las siguientes preguntas

- ¿Quiénes deben participar?
- ¿Qué deben hacer?
- ¿Cuándo lo deben hacer?
- ¿Cómo lo deben hacer?

Elementos adicionales del Proceso

- Revisiones y Auditorías Realizadas.
- Productos de trabajo que deben ser administrados y controlados (o puesto bajo gestión de configuración).
- Mediciones que deben realizarse.
- Capacitación.
- Herramientas

Principales Modelos de Calidad existentes

ISO
Un conjunto de estándares internacionales que se pueden usar en el desarrollo de un sistema de gestión de calidad en todas las industrias, es el ISO 9000. Los estándares ISO 9000 pueden aplicarse a un amplio abanico de organizaciones desde las de manufactura hasta las de servicios. ISO 9001 es el mas general de estos estándares y se aplica en organizaciones interesadas en el proceso de calidad de diseño, desarrollo y mantenimiento de productos.
ISO9001 no es un estándar específico para el desarrollo de software, pero define principios generales que pueden aplicarse al software. El estándar ISO 9001 describe varios aspectos del proceso de calidad y define que estándares y procedimientos deben existir en una organización. Estos deben documentarse en un manual de calidad organizacional. La definición del proceso debe incluir una descripción de la documentación requerida, donde se demuestre que los procesos definidos han sido seguidos durante el desarrollo del producto.
Los estándares ISO 9001 no definen los procesos de calidad que deben usarse. De hecho no restringe los procesos usados en una organización de ninguna forma. Esto permite flexibilidad en todos los sectores industriales e implica que pequeñas compañías puedan tener procesos no burocráticos y que cumplan con la normativa ISO 9000.
Sin embargo esta flexibilidad implica que no podemos hacer suposiciones a cerca de el parecido o diferencia de los procesos de compañias que cumplen con la normativa ISO 9000.
Los principales beneficios de seguir ISO son:

- Mejorar la satisfacción del cliente
- Mejorar continuamente los procesos relacionados con la Calidad.

Otros beneficios adicionales son:

- Reducción de rechazos e incidencias en la producción o prestación del servicio
- Aumento de la productividad

Algunas personas piensan que la certificación ISO 9000 significa que la calidad del software producido por las compañías certificadas será mejor que la de las compañías no certificadas. Este no es ciertamente el caso. El estándar ISO 9000 se refiere simplemente a la definición de los procedimientos que utilizados en la compañía y la documentación asociada que muestre que los procesos han sido seguidos. Este no se ocupa de asegurar que estos procesos sean la mejor práctica, ni asegura la calidad el producto.

CMMI
El modelo CMMI intenta ser un marco de trabajo para la mejora del proceso que sea aplicable en un amplio abanico de compañía. Su versión en etapas es compatible con el CMM de software y permite un desarrollo del sistema de la organización, la gestión procesos a valorar y su asignación a un nivel de madurez entre 1 y 5. Su versión continua permite una clasificación mas detallada y considera 24 ares de procesos, en una escala de 1 a 6.

- *Área de proceso*: El CMMI identifica 24 areas de procesos que son relevantes para la capacidad y la mejora del proceso de software.

- *Metas*: Las metas son descripciones abstractas de un estado deseable que debería ser alcanzado por una organización. El CMMI tiene metas especificas con cada área de proceso.
- *Practicas*: Las prácticas en el CMMI son descripciones de vias para conseguir una meta.

La valoración de un CMMI implica examinar los procesos en una organización y clasificarlos en una escala de seis puntos que reflejan el nivel de madurez en cada area de proceso.
CMMI es uno de los modelos más implementado en todo el mundo.
No es una norma, y no se "certifica", sólo se evalúa a través de profesionales reconocidos por el SEI como *Lead Appraisers*

Por etapas

- 5 Niveles, de 1 a 5
- Definidos por un conjunto de Áreas de Proceso.
- Niveles indican "Madurez Organizacional"
- Similar al SW-CMM
- Provee una única clasificación que facilita comparaciones entre organizaciones.
- Provee una secuencia probada de mejoras.

Continuo

- 6 Niveles de 0 a 5
- Definidos por cada Áreas de Proceso.
- Niveles indican "Capacidad" de un Área de Proceso.
- Similar al EIA/IS-731
- Permite comparaciones sobre la base de cada AP.
- Permite elegir el orden de las mejoras.

SPICE
Es un modelo que aproxima a la valoración de la capacidad y a la mejora del proceso, es más flexible que el modelo CMMI. Incluye niveles de madurez similares a los niveles del CMMI pero además identifica procesos, como los procesos entre cliente y proveedor que recorren estos niveles. A medida que subimos en nivel de madure, el rendimiento de estos procesos claves debe mejorarse.

<u>Auditoría Interna y Auditoría Externa</u>

La auditoría interna es la realizada con recursos materiales y personas que pertenecen a la empresa auditada. Los empleados que realizan esta tarea son remunerados económicamente. La auditoría interna existe por expresa decisión de la Empresa, o sea, que puede optar por su disolución en cualquier momento.

Por otro lado, la auditoría externa es realizada por personas afines a la empresa auditada; es siempre remunerada. Se presupone una mayor objetividad que en la Auditoría Interna, debido al mayor distanciamiento entre auditores y auditados.

La auditoría informática interna cuenta con algunas ventajas adicionales muy importantes respecto de la auditoría externa, las cuales no son tan perceptibles como en las auditorías convencionales. La auditoría interna tiene la ventaja de que puede actuar periódicamente realizando Revisiones globales, como parte de su Plan Anual y de su actividad normal. Los auditados conocen estos planes y se habitúan a las Auditorías, especialmente cuando las consecuencias de las Recomendaciones habidas benefician su trabajo.

Aunque la auditoría interna sea independiente del Departamento de Sistemas, sigue siendo la misma empresa, por lo tanto, es necesario que se le realicen auditorías externas como para tener una visión desde afuera de la empresa.

La auditoría informática, tanto externa como interna, debe ser una actividad exenta de cualquier contenido o matiz "político" ajeno a la propia estrategia y política general de la empresa. La función auditora puede actuar de oficio, por iniciativa del propio órgano, o a instancias de parte, esto es, por encargo de la dirección o cliente.

¿Por qué auditar?

- Porque se da una opinión objetiva e independiente
- Porque permite identificar áreas de insatisfacción potencial del cliente
- Porque nos permite asegurar al cliente que estamos cumpliendo con nuestras expectativas
- Porque permite identificar oportunidades de mejora.

Beneficios de las auditorías de calidad de software

- Permiten evaluar el cumplimiento del proceso de desarrollo
- Permiten determinar la implementación efectiva de:
    - El proceso de desarrollo organizacional
    - El proceso de desarrollo del proyecto
    - Las actividades de soporte
- Proveen mayor visibilidad a la gerencia sobre los procesos de trabajo
- Los resultados de las auditorias solicitando acciones correctivas conllevan a la mejora del proceso y del producto

Tipos de auditorias de calidad de software

Auditorías de Proyecto

- El objetivo de esta auditoría es verificar objetivamente la consistencia del producto a medida que evoluciona a lo largo del proceso de desarrollo, determinando que:
    - Las interfaces de hardware y software sean consistentes con los requerimientos de diseño en la ERS.
    - Los requerimientos funcionales de la ERS se validan en el Plan De Verificación y Validación de Sotware.
    - El diseño del producto, a medida que DDS evoluciona, satisface los requerimientos funcionales de la ERS.
    - El código es consistente con el DDS.

Auditoría Funcional

- La auditoría funcional compara el software que se ha construido (incluyendo sus formas ejecutables y su documentación disponible) con los requerimientos de software especificados en la ERS.
- El propósito de la auditoría funcional es asegurar que el código implementa sólo y completamente los requerimientos y las capacidades funcionales descriptos en la ERS.
- El responsable de QA deberá validar si la matriz de rastreabilidad está actualizada.

Auditoría Física

- La auditoría física compara el código con la documentación de soporte.
- Su propósito es asegurar que la documentación que se entregará es consistente y describe correctamente al código desarrollado.
- El PACS debería indicar la persona responsable de realizar la auditoría física.
- El software podrá entregarse sólo cuando se hayan arreglado las desviaciones encontradas.

Responsabilidades

- Gerente de SQA:
  - prepara el plan de auditorias,
  - calcula el costo de las auditorias
  - asigna los recursos.
  - responsable de resolver las no-conformidades
- Auditor:
  - acuerda la fecha de la auditoria,
  - comunica el alcance de la auditoria,
  - recolecta y analiza la evidencia objetiva que es relevante y suficiente para tomar conclusiones
  - acerca del proyecto auditado,
  - realiza la auditoria,
  - prepara el reporte,
  - realiza el seguimiento de los planes de acción acordados con el auditado
- Auditado:
  - acuerda la fecha de la auditoria,
  - participa de la auditoria,
  - proporciona evidencia al auditor.
  - contesta al reporte de auditoria,
  - propone el plan de acción para deficiencias citadas en el reporte
  - comunica el cumplimiento del plan de acción.