INTRODUCCIÓN

La ingeniería del software es una disciplina de la ingeniería cuya meta es el desarrollo costeable de sistemas de software. Éste es abstracto, intangible y fácil de modificar. Que el software tenga estas características simplifica la ingeniería del software ya que no existen limitaciones físicas del potencial del software, pero la dificulta en el sentido que el software puede llegar a ser extremadamente complejo y difícil de entender, debido a no tener restricciones naturales.

La noción de **ingeniería del software** surgió en el año 1968, debido a la "**crisis del software**".

CRISIS DEL SOFTWARE	
CAUSA	Introducción de nuevas computadoras hardware basadas en circuitos integrados. Esto provocó la posibilidad de desarrollar software más grande y más complejo.
SINTOMAS	• Los proyectos se atrasaban. • Se sobrepasaban los presupuestos. • Muchos proyectos se cancelaban. • Se comenzaba a desarrollar software sin especificaciones.
CONSECUENCIAS	Software de mala calidad y de desempeño pobre, lo cual derivaba en la realización de intensas actividades de mantenimiento, lo cual degrada el software.

Entonces fue evidente que para crear software de esta magnitud tomar un enfoque informal no era lo adecuado. Se necesitaban nuevas técnicas y métodos para controlar la complejidad inherente a los sistemas grandes. Sin embargo, cuanto más crezca nuestra capacidad para producir software, también lo hará la complejidad de los sistemas de software solicitados.

Software

Un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios web que permitan a los usuarios descargar la información de productos recientes. Existen 2 tipos:

- **Productos genéricos:** Son sistemas aislados producidos por una organización de desarrollo y que se venden al mercado abierto a cualquier cliente. La especificación es controlada por la organización que desarrolla.
- **Productos personalizados (o hechos a medida):** Son sistemas requeridos por un cliente en particular. La especificación de los productos personalizados, por lo general, es desarrollada y controlada por la organización que compra el software.

Ingeniería del Software

La ingeniería del software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después de que se utiliza. En esta definición, existen dos frases clave:

- **Disciplina de la ingeniería:** Los ingenieros aplican teorías, métodos y herramientas donde sean convenientes, pero las utilizan de forma selectiva y siempre tratando de descubrir soluciones a los problemas, aun cuando no existan teorías y métodos aplicables para resolverlos. Los ingenieros también saben que deben trabajar con restricciones financieras y organizacionales, por lo que buscan soluciones tomando en cuenta estas restricciones.
- **Todos los aspectos de producción de software**: La ingeniería comprende disciplinas:
  - **Técnicas:** Ayudan a construir el producto. Éstas son: Rq´s, análisis, diseño, implementación, prueba.
  - **De Gestión:** Planificación de proyectos, Monitoreo y control.
  - **De Soporte:** Gestión de Configuración, métricas, aseguramiento de la calidad.

Proceso de Software

Es un conjunto de actividades y resultados asociados que producen un producto de software. Existen cuatro actividades fundamentales que son comunes para todos los procesos y son:

- **Especificación de Software:** Los clientes e ingenieros definen el software a producir y las restricciones sobre su operación.
- **Desarrollo de Software:** El software se diseña y programa.
- **Validación de Software:** El software se valida para asegurar que es lo que el cliente realmente quiere.
- **Evolución del Software:** El software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado.

El proceso de desarrollo se elegirá de acuerdo al tipo de sistema que se vaya a desarrollar. Además estas actividades genéricas pueden organizarse de diferentes formas y describirse en diferentes niveles de detalle para diferentes tipos de software. Sin embargo, el uso de un proceso inadecuado del software puede reducir la calidad o la utilidad del producto de software que se va a desarrollar y/o incrementar los costos de desarrollo.

Modelo de procesos del software

Es una descripción simplificada de un proceso del software que presenta una visión de ese proceso. Estos modelos pueden incluir:

- Actividades.
- Productos.
- Papel de las personas involucradas.

Algunos ejemplos de estos tipos de modelos son los modelos de flujo de trabajo, modelos de flujo de datos o de actividad, y modelos de rol acción.

Estos modelos se basan en uno de los tres modelos generales o paradigmas de desarrollo de software:

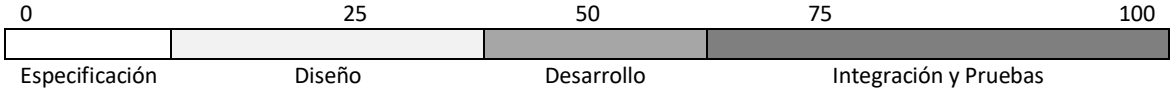
- **Enfoque Cascada:** Considera las actividades anteriores y las representa como fases de procesos separados. Después de que cada etapa queda definida «se firma» y el desarrollo continúa con la siguiente etapa.
- **Desarrollo iterativo:** Este enfoque entrelaza las actividades de especificación, desarrollo y validación. Un sistema inicial se desarrolla rápidamente a partir de especificaciones muy abstractas y luego se refina basándose en las peticiones del cliente, de manera de producir un sistema que satisfaga sus necesidades.
- **Ingeniería del software basada en componentes (CBSE):** Esta técnica supone que las partes del sistema ya existen, por lo cual se enfoca en la integración de estas partes más que en desarrollarlas desde el principio.

Costos de la ingeniería del software

La distribución de costos a través de las diferentes actividades en el proceso del software depende del proceso utilizado y del tipo de software que se vaya a desarrollar.

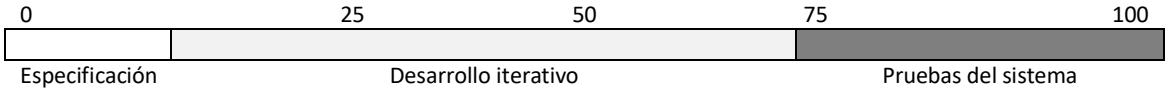
Costos del desarrollo para sistemas a medida:

- **Utilizando un enfoque en cascada:** Los costos de especificación, diseño, implementación e integración se miden de forma separada. La integración y pruebas del sistema son las actividades de desarrollo más caras. Normalmente, suponen alrededor del 40%, 50% o más del costo del desarrollo total.

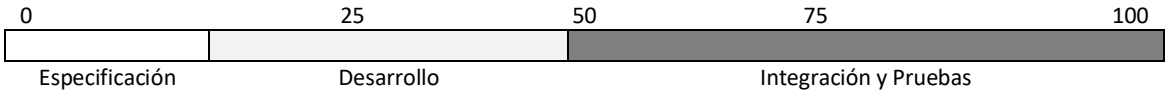


- **Utilizando un enfoque iterativo:** Los costos de la especificación se reducen debido a que sólo se produce la especificación de alto nivel antes que el desarrollo. La especificación, el diseño, la implementación, la integración y las pruebas se llevan a cabo en paralelo dentro de una actividad

de desarrollo. Sin embargo, aún se necesita una actividad independiente de pruebas del sistema una vez que la implementación inicial esté completa.



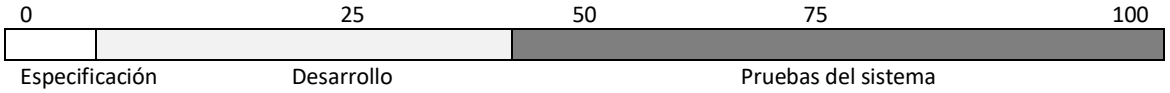
- **Utilizando ingeniería del software basada en componentes:** Los costos de desarrollo se reducen en relación a los costos de integración y pruebas. Los costos de integración y pruebas se incrementan porque tenemos que asegurarnos de que los componentes que utilizamos cumplen realmente su especificación y funcionan como se espera.



Costos de evolución para sistemas a medida:

- Para sistemas software de larga vida, que pueden ser usados durante 10 años o más, estos costos exceden a los de desarrollo por un factor de 3 o 4.
- Para sistemas de negocio más pequeños, que tienen una vida mucho más corta, los costos de evolución son más reducidos.

Costos del desarrollo para sistemas genéricos: Los costos de la especificación son relativamente bajos. Sin embargo, debido que se pretende utilizarlos en diferentes configuraciones, deben ser probados a fondo.



Costos de evolución para sistemas a medida: En muchos casos, existe poca evolución de un producto. Una vez que una versión de producto se entrega, se inicia el trabajo para entregar la siguiente y, por razones de mercadotecnia, ésta se presenta como un producto nuevo más que como una versión modificada de un producto que el usuario ya adquirió. Por lo tanto, los costos de evolución no se consideran de forma separada, sino que son sencillamente los costos del desarrollo para la siguiente versión del sistema.

Métodos de la ingeniería del software

Es un enfoque estructurado para el desarrollo de software cuyo propósito es facilitar la producción de software de alta calidad de una forma costeable. Algunos de ellos son:

- Métodos orientados a funciones: Por ejemplo Análisis Estructurado y JSD.
- Métodos orientados a objetos.

No existe un método ideal, y métodos diferentes tienen distintas áreas donde son aplicables.
Todos los métodos se basan en la idea de modelos gráficos de desarrollo de un sistema y en el uso de estos modelos como un sistema de especificación o diseño. Los métodos incluyen varios componentes diferentes.

Atributos de un buen software

Los atributos reflejan la calidad del software. No están directamente asociados con lo que el software hace. Más bien, reflejan su comportamiento durante su ejecución y en la estructura y organización del programa fuente y en la documentación asociada.
El conjunto específico de atributos que se espera de un sistema de software depende obviamente de su aplicación.

ATRIBUTOS ESENCIALES DE UN BUEN SOFTWARE	
MANTENIBILIDAD	El software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de Cambio de los dientes. Éste es un atributo crítico debido a que el

	cambio en el software es una consecuencia inevitable de un cambio en el entorno de negocios.
CONFIABILIDAD	La confiabilidad del software tiene un gran número de características, incluyendo la fiabilidad, protección y seguridad. El software confiable no debe causar daños físicos o económicos en el caso de una falla del sistema.
EFICIENCIA	El software no debe hacer que se malgasten los recursos del sistema, como la memoria y los ciclos de procesamiento. La eficiencia incluye tiempos de respuesta y de procesamiento, utilización de memoria, etc.
USABILIDAD	El software debe ser fácil de utilizar, sin esfuerzo adicional, por el usuario para quien está diseñado. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada.

Retos que afronta la ingeniería del software

- **Reto de la heterogeneidad:** Consiste en desarrollar técnicas para construir software confiable que sea lo suficientemente flexible, es decir, que pueda operar como un sistema distribuido en redes con distintos tipos de computadoras, que se integre con sistemas heredados, con sistemas desarrollados en otros lenguajes, etc.
- **Reto de la entrega:** Reducir los tiempos de entrega para sistemas grandes y complejos sin comprometer la calidad del sistema.
- **Reto de la confianza:** Desarrollar técnicas que demuestren que los usuarios pueden confiar en el software.

GESTIÓN DE PROYECTOS

Características de un Proyecto

- Están orientados a 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.
  - Los objetivos deben ser claros y alcanzables.
- Tienen una duración limitada en el tiempo, tienen principio y fin:
  - Los proyectos son temporarios, cuando se alcanzan los objetivos, el proyecto termina.
  - Una línea de producción no es un proyecto.
- Implican tareas interrelacionadas basadas en esfuerzos y recursos.
- Son únicos: Todos los proyectos por similares que sean tienen características que los hacen únicos, por ejemplo, que tienen cronogramas diferentes.

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).

Factores de Calidad

- Objetivos de proyecto.
- Tiempo.
- Costos.

El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado. Es responsabilidad del Líder de proyecto balancear estos tres objetivos (que a menudo compiten entre ellos y por lo general casi nunca se cumplen simultáneamente).

ERRORES CLASICOS EN PROYECTOS Y ORGANIZACIONES

El desarrollo de software es una actividad complicada. Un proyecto de software típico puede presentar muchas oportunidades para aprender de lo errores originados en la aplicación frecuente de prácticas inefectivas. Estos errores se clasifican en:

- Gente.
- Proceso.
- Producto.
- Tecnología.

Gente

1. **Motivación débil:** Se ha demostrado que la motivación tiene mayor efecto sobre la productividad y la calidad que ningún otro factor.
2. **Personal mediocre:** La capacidad individual de los miembros del equipo, así como sus relaciones como equipo, probablemente tienen la mayor influencia en la productividad.
3. **Empleados problemáticos incontrolados:** Amenazan la velocidad del desarrollo.
4. **Hazañas:** Algunos desarrolladores ponen un gran énfasis en la realización de hazañas en sus proyectos. Pero lo que hacen tienen más de malo que de bueno. Este tipo de actitudes convierten pequeños contratiempos en auténticos desastres.
5. **Añadir mas personal a un proyecto retrasado:** Es este el más clásico de los errores. El aprendizaje agrega más tiempo.
6. **Oficinas repletas y ruidosas:** La mayoría de los desarrolladores consideran sus condiciones de trabajo como insatisfactorias. Los entornos repletos y ruidosos alargan los planes de desarrollo.
7. **Fricciones entre clientes y desarrolladores:** El principal efecto de esta fricción es la mala comunicación y los efectos secundarios son pobres entendimientos de los requerimientos, pobre desarrollo de la interfaz, rechazo del cliente por el producto terminado.
8. **Expectativas poco realistas:** Una de las causas más comunes de fricciones entre los desarrolladores y sus clientes o los directivos son las expectativas poco realistas.
9. **Falta de un promotor efectivo del proyecto:** Sin un promotor ejecutivo del proyecto el resto de los gerentes de la empresa puede presionar para que se acepten fechas poco realistas o hacer cambios que afecten seriamente al proyecto.
10. **Falta de la participación de los implicados.**
11. **Falta de la participación del usuario.**
12. **Política antes que desarrollo:** Primar la política en vez de los resultados. Es fatal para el desarrollo orientado a la velocidad.
13. **Ilusiones:** Las ilusiones no son sólo optimismo. Realmente consisten en cerrar los ojos y esperar que todo funcione cuando no se tienen las bases razonables para pensar que será así. Impiden llevar a cabo una planificación coherente y pueden ser la raíz de más problemas en el software que todas las otras causas combinadas.

Proceso

Planificación excesivamente optimista: Esto hace que el proyecto falle por infravalorar el alcance del proyecto, reduciendo las actividades críticas. También afecta a los desarrolladores, quienes se ven afectados en su moral y productividad.

14. **Gestión de riesgos insuficientes:** No hacerlo ya es un error clásico y muchos de los errores resaltados aquí son consecuencia de no llevar a cabo esta etapa en un proyecto.
15. **Fallos de los contratados:** Si las gestiones con terceros no se gestionan oportunamente, la utilización de contratados puede demorar el proyecto en vez de acelerarlo.
16. **Planificación Insuficiente:** Sino se planifica un desarrollo rápido no se puede esperar obtenerlo.
17. **Abandonar la planificación bajo presión:** Los equipos desarrollan planes y los abandonan cuando tropiezan con un problema de planificación. El problema no está en el abandono del plan sino en no tener un plan alternativo y entonces se cae en el modo de codificar y corregir.
18. **Pérdida de tiempo en el inicio difuso:** Es el tiempo que transcurre antes del inicio del proyecto.

19. **Escatimar las actividades iniciales:** Muchos proyectos se aceleran intentando acotar las actividades no esenciales, y por lo general estas actividades son el análisis de requerimientos, arquitectura y diseño, ya que no generan código. Los resultados de este error también son conocidos como "saltar a la codificación" y son predecibles los resultados.
20. **Diseño Inadecuado.**
21. **Escatimar control de calidad**: Proyectos apurados recortan las revisiones de código, los testeos y las revisiones de diseño. Acortar un día estas actividades presupone de 3 a 10 días de retraso al final del proyecto.
22. **Control Insuficiente de la Dirección**.
23. **Convergencia prematura o excesivamente frecuente:** Bastante antes de entregar el producto hay un esfuerzo por entregarlo y entonces se preparan varias actividades que irían al final, esto quita tiempo a otras actividades y no beneficia en nada, puesto que el producto todavía esta siendo desarrollado.
24. **Omitir tareas necesarias en la estimación:** Sino se mantienen registro de anteriores proyectos, se olvidan las tareas menos visibles y son las que luego agregan tiempos. Esfuerzo omitido en un plan agrega de un 20% a 30% al esfuerzo de desarrollo.
25. **Planificar ponerse al día mas adelante.**
26. **Programación a Destajo.** Algunas organizaciones creen que la programación rápida, libre, tal como salga es el camino hacia el desarrollo rápido. Esto rara vez funciona.

Producto

27. **Exceso de Requerimientos:** Algunos proyectos tienen más requerimientos de los que necesitan, desde el mismo inicio, lo que alarga innecesariamente el plan de proyecto.
28. **Cambio de las prestaciones:** Un cambio de este calibre pude producir un aumento en el plan de un 25% o más, lo que puede ser fatal para un proyecto de desarrollo rápido.
29. **Desarrolladores meticulosos:** Los desarrollares a menudo se muestran fascinados por una nueva tecnología y quieren implantar estas nuevas características sea que el producto las necesite o no. El esfuerzo requerido por diseñar, codificar, testear y documentar estas prestaciones alarga el plan.
30. **Tires y aflojes en la negociación:** Un directivo que aprueba un retraso en el plan lo hace sabiendo que el plan estaba equivocado, pero una vez que lo corrige realiza acciones explícitas para volver a equivocarse.
31. **Desarrollo orientado a la investigación:** Si el proyecto fuerza los límites de la informática porque necesita la creación de nuevos algoritmos o de nuevas técnicas de computación, no estamos desarrollando software, estamos investigando. Los planes de desarrollo son razonablemente predecibles, los planes en la investigación sobre software ni siquiera son predecibles teóricamente.

Tecnología

32. **Síndrome Silver-bullet (de las balas de plata):** Cuando el equipo del proyecto se aferra sólo a una nueva técnica, una nueva tecnología o un proceso rígido y espera resolver con ello sus problemas de planificación, está inevitablemente equivocado.
33. **Sobreestimación de las ventajas del empleo de nuevas herramientas o métodos:** Las organizaciones mejoran raramente su productividad a grandes saltos, sin importar cuantas nuevas herramientas o método empleen. Los beneficios de las nuevas técnicas son parcialmente desplazados por las curvas de aprendizaje y también se suponen nuevos riesgos que solo se descubren utilizando dichas técnicas.
34. **Cambiar de herramientas a mitad del proyecto:** Es un viejo recurso que funciona raramente. Quizás convenga cambiar de la versión 3 a la 3.1 o quizás a la 4. Pero la curva de aprendizaje, el retrabajo y los errores cometidos con la nueva herramienta, normalmente anulan cualquier beneficio.
35. **Falta de control automático de código fuente:** Sin él se expone el proyecto a riesgos innecesarios y dos programadores que trabajan sobre un mismo trozo de código deben coordinar manualmente sobre que versión se hacen las actualizaciones.

Habilidades (skills) de un Líder de Proyecto

- Sentirte cómodo con los cambios.
- Entender a la organización y su estructura.

- Ser capaz de guiar al equipo y motivarlo para satisfacer los objetivos del proyecto.
- Necesidad de dos tipos de skills:
    - Hard skills: Conocimiento del producto, conocer las herramientas y técnicas de administración de proyectos.
    - Soft skills: Ser capaz de trabajar con gente. Esto incluye:
        - Comunicación: saber escuchar, persuadir.
        - Organizacional: Planificar, plantear de objetivos, análisis.
        - Team Building: Empatía, motivación, espíritu de cuerpo.
        - Liderazgo: Dar ejemplos, enérgico, tener el big picture, delegar, ser positivo.
        - Flexibilidad, creatividad, paciencia, persistencia.
        - Tecnológicos: experiencia, conocimiento del proyecto.

ETAPAS DEL PROYECTO

- Iniciación del proyecto.
- Planificación del proyecto y armado del equipo.
- Ejecución.
- Tracking y control.
- Post mortem.

Antes del proyecto

Antes de iniciar un proyecto se debe hacer:

- **Identificación del cliente y del mercado:** Es decir, identificar el dominio, el soporte que se debe proveer, si se debe cumplir con algún marco de calidad y las líneas de reporte.
- **Estudio de factibilidad:** Es un estudio que se realiza a fin de verificar si el proyecto puede realizarse o no. Se llevan a cabo 3 tipos de estudios (factibilidad técnica, económica y operativa). Se evalúa si el proyecto realmente se necesita, cuáles son los requerimientos iniciales, si se dispone del capital necesario y además si el proyecto funcionará y garantizará un éxito mínimo.
- **Propuesta de Proyecto:**
    - Resumen ejecutivo: Cuáles son las funciones básicas y los beneficios al cliente.
    - Arquitectura del sistema: Es decir, qué se va a construir.
    - Plan de Proyecto: Cuáles son los entregables y sus fechas de entrega.
    - Costos: Se pueden dividir por perfiles de los empleados.
- **Negociaciones:**
    - Características del producto: Funcionalidad más importante que tiene el producto. Las características deben estar priorizadas, ser realistas, claramente entendidas y medibles.
    - Cronograma o Schedule: Determinar fechas de comienzo y fin, milestones o Hitos, entregas de productos.
    - Presupuesto: Del Staff, del software y hardware.
    - Estructura del equipo: Curva de Staffing, cantidad de personas por rol.
    - Consideraciones: Si los riesgos se van a compartir, quién será el dueño del código fuente, etc.

1- Inicio del Proyecto

Debo Elegir un nombre para el proyecto y anunciarlo, esto dará inicio oficial al proyecto. Se debe definir:

- Estructura del equipo.
- Necesidades de soporte.
- Necesidades financieras.
- Objetivos de calidad.

2- Planificación del proyecto y armado del equipo.

- **Definición del proyecto:** Determinar los problemas, las oportunidades, la misión, los objetivos. Es importante que éstos últimos sean:
    - Críticos.
    - Observables.
    - Distinguibles.
    - Medibles.

Es importante también definir las asunciones, los riesgos y las condiciones de aceptación del producto. Esto último permite marcar cuándo el proyecto de desarrollo termina, y cuándo comienza otro de mantenimiento, es decir, marcar los límites del proyecto.

- **Creación del plan:** El **Plan de Proyecto** fija los recursos disponibles, divide el trabajo y crea el calendario de trabajo. Debe utilizarse como un conductor para el proyecto y debe ser lo mejor posible de acuerdo a la información de la que se dispone. Este plan provee una herramienta de comunicación estándar a través del ciclo de vida de un proyecto.
  La planificación es un proceso iterativo, que solamente se completa cuando el proyecto se termina. Conforma la información se hace disponible, el plan debe revisarse y adaptarse.

- **Infraestructura del proyecto:** Los proyectos necesitan empezar con una infraestructura básica, que puede incluir:
  - Administración de Configuración.
  - Un directorio para el proyecto.
  - Lista de Mails.
  - Sitio Web para realizar la comunicación, por ejemplo para postear novedades de l proyecto.
  - Herramientas varias, como ser los entornos de desarrollo y testing.

- **Formación del equipo:** Determinar quiénes serán los miembros iniciales del proyecto, los que provienen de otros proyectos, y cuáles serán sus roles.

- **Actividades Principales:**
  - Armar la infraestructura del proyecto y su entorno.
  - Asegurar el entrenamiento que haga falta: Capacitación General, Capacitación basada en roles específicos, Capacitación de dominios específicos.
  - Producir los documentos "claves" del proyecto:
    - ✓ **REQB**: Libro de requerimientos (Una ERS por ejemplo)
    - ✓ **SPMP:** Plan de Proyecto.
    - ✓ **SQAP** Plan de Aseguramiento de Calidad: Describe los procedimientos y estándares de calidad que se utilizarán en el proyecto.
    - ✓ **SCMP** Plan de Gestión de Configuración: Describe los procedimientos para la gestión de configuraciones y su estructura.
    - ✓ **Directorio del proyecto o repositorio.**

WORK BREAKDOWN STRUCTURE (WBS)

Una WBS o Estructura de Desglose de Trabajo identifica las actividades que necesitan ser realizadas durante el proyecto. Sus objetivos son:

- Desglosar hasta llegar al nivel de actividades. El límite es que cada actividad debe producir un "entregable" (código, modelos).
- Hacer un bosquejo de cómo se van a estructurar las actividades:
  - En partes lógicas.
  - De acuerdo a las etapas del proyecto.
  - De acuerdo a los paquetes o módulos, etc.

El líder de proyecto es quien decide sobre el enfoque de arquitectura y el nivel de detalle de la WBS. A veces se sigue el siguiente criterio:

- Organizar por etapas → Para empresas que no desarrollan software, o se si va a tercerizar alguna etapa.
- Organizar por módulos → Conviene si hay grupos de trabajo por módulo, o cuando algunos módulos tienen que estar listos antes que otros.

Se dice que una WBS está lista cuando:

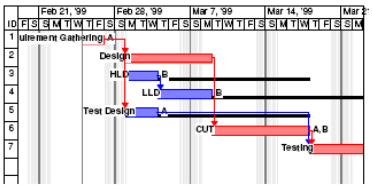
- Los elementos del WBS (tareas) son medibles.
- Los eventos que marcan el Comienzo / Fin están claramente definidos.

- Cada “actividad” tiene un entregable.
- Tiempos / costos pueden ser estimados.
- La duración de las actividades tienen limites aceptables (por lo general se recomiendan actividades con duración mínima de 1 semana pero no mayor a las 8 semanas).
- Las asignaciones pueden ser hechas independientemente.

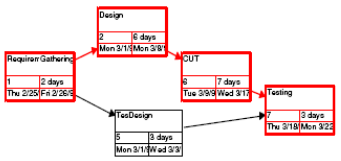
SCHEDULE

Determina qué tareas deben ser hechas secuencialmente, cuáles pueden ser hechas en paralelo, y luego estima el tiempo de realización de las tareas. Debe tenerse en cuenta también la asignación de recursos a cada una de las actividades.
Un problema en la calendarización surge cuando no se posee información acerca de la duración de actividades similares de proyectos anteriores, lo cual hace que se posea cierta incertidumbre al realizar dichas estimaciones.
Al Schedule podemos representarlo a través de 2 gráficos:

- **Grafico de Gantt:** Es un gráfico de barras, cuyo largo es proporcional a la duración de cada actividad. Es la forma más fácil e intuitiva para visualizar un Schedule y es útil para el tracking.



- **Grafico de Pert:** Es un diagrama de red, que muestra todas las actividades y sus dependencias y hace evidente el camino crítico. Es útil para comprender cuando un set de tareas debe ser completado, pero no es bueno para el tracking.



El camino crítico esta constituido por aquellas actividades que no pueden atrasarse sin aumentar el tiempo total del proyecto.

HITOS Y ENTREGAS

Un hito es el punto final de una actividad del proceso de software. Cuando se planifica un proyecto, se debe establecer una serie de **hitos.** Éstos deben representar el fin de una etapa lógica en el proyecto y para cada uno de ellos debe existir una salida formal. Una entrega es el resultado del proyecto que se entrega al cliente al final de una fase principal del proyecto. Como regla general, las entregas son hitos, pero éstos no necesariamente son entregas. Dichos hitos pueden ser resultados internos del proyecto que son utilizados por el gestor del proyecto para verificar el progreso del mismo pero que no se entregan al cliente.

GESTIÓN DE RIESGOS

Es el proceso de identificación de riesgos y manejo de aquellos más importantes para el proyecto de forma tal que no se conviertan en problemas, afectando a la programación del proyecto o a la calidad del software a desarrollar.
El proceso de gestión de riesgos es un proceso iterativo, que se aplica a lo largo de todo el proyecto. A medida que se tenga más información, los riesgos deben analizarse y priorizarse, y deben modificarse los planes de mitigación y contingencia si fuera necesario.

Riesgo: Evento que podría comprometer el éxito del proyecto. Probabilidad de que una circunstancia adversa ocurra.

- Toda actividad lleva implícita un riesgo.
- El riesgo acompaña a todo cambio.
- El riesgo implica elección e incertidumbre.
- Los riesgos están caracterizados por la incertidumbre.

Es importante saber diferenciar los riesgos, de los problemas. Los riesgos tienen condiciones y consecuencias inciertas y pueden ser dinámicos. Los problemas son ciertos.

Categorías de los Riesgos

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

Los riesgos que pueden afectar a un proyecto dependen del propio proyecto y del entorno organizacional donde se desarrolla. Sin embargo, muchos riesgos son universales.
La gestión de riesgos es importante particularmente para los proyectos de software debido a las incertidumbres inherentes con las que se enfrentan muchos proyectos.

Estrategias frente a los riesgos

- **Estrategias reactivas:** Se evalúan las consecuencias del riesgo cuando éste ya ha ocurrido. Esto pone en peligro el proyecto.
- **Estrategias proactivas:** Se caracteriza por una evaluación previa y sistemática de riesgos y sus consecuencias, creando planes de mitigación (para reducir el efecto o la probabilidad del riesgo) y planes de contingencia.

Etapas de la Gestión de Riesgos

Identificación de riesgos
Comprende el descubrimiento de los posibles riesgos del proyecto. En esta etapa no hay que valorarlos o darles prioridad, aunque por lo general no se consideran los riesgos con consecuencias menores o con baja probabilidad. Tipos de riesgos (o categorías):

- **Riesgos de tecnología:** Se derivan de las tecnologías de software o de hardware utilizadas en el sistema que se está desarrollando.
- **Riesgos de personal (o relacionados con la experiencia y tamaño del equipo):** Riesgos asociados con las personas del equipo de desarrollo.
- **Riesgos organizacionales (o relacionados con el impacto en la organización):** Se derivan del entorno organizacional donde el software se está desarrollando.
- **Riesgos de herramientas (o Relacionados con el entorno de desarrollo):** Se derivan de herramientas CASE y de otro software de apoyo utilizado para desarrollar el sistema.
- **Riesgos de requerimientos:** Se derivan de los cambios de los requerimientos del cliente y el proceso de gestionar dicho cambio.
- **Riesgos de estimación:** Se derivan de los estimados administrativos de las características del sistema y los recursos requeridos para construir dicho sistema.

Otros (estaban en la filmina):

- **Asociados con el tamaño del producto.**
- **Relacionados con el cliente.**
- **Asociados a la definición del proceso de producción.**

Análisis de Riesgos
Durante este proceso se considera por separado cada riesgo identificado y se decide acerca de la probabilidad y el impacto del mismo. No se hace una valoración con números precisos sino en intervalos. La medición se realiza de la siguiente manera:

- La probabilidad del riesgo se puede valorar como muy bajo « 10%), bajo (10-25%), moderado (25-50%), alto (50-75%) o muy alto (>75%).
- El impacto del riesgo puede ser valorado como catastrófico, serio, tolerable o insignificante. El impacto del riesgo se mide de acuerdo a 2 aspectos:
    - Alcance: Cuán serio es y cuánto del proyecto se ve afectado.

    - Temporalización de los efectos: Cuándo y por cuánto tiempo.
- La exposición se calcula como el producto entre la probabilidad y el impacto. Representa a la amenaza total del riesgo.

Luego se colocan los riesgos en una tabla ordenados de acuerdo a la exposición de los mismos. Como no pueden gestionarse absolutamente todos los riesgos asociados al proyecto, ya que se necesitaría demasiada información, es necesario discernir cuáles son los más importantes que se deben tratar (se recomiendan 10, pero este numero puede variar). Por lo general se desprecian riesgos poco probables y los medianamente probables con poco impacto.

Planificación de Riesgos

En esta etapa se definen las estrategias para gestionar cada riesgo, las cuales pueden dividirse en tres categorías:

- **Estrategias de prevención:** Están orientadas a reducir la probabilidad del riesgo.
- **Estrategias de minimización:** Intentan reducir el impacto del riesgo.
- **Planes de contingencia:** Definen cuál es el plan a seguir en el caso de que el riesgo ocurra. A veces se implementan porque es más barato solucionar el problema que evitarlo.

Supervisión de Riesgos

En esta etapa se valora cada uno de los riesgos identificados para decidir si éste es más o menos probable y si han cambiado sus efectos. La supervisión de riesgos debe ser un proceso continuo y, en cada revisión, cada uno de los riesgos debe ser analizado por separado ya que éstos pueden desaparecer, ocurrir o mantenerse, y también pueden aparecer nuevos riesgos.

Estimación de riesgos

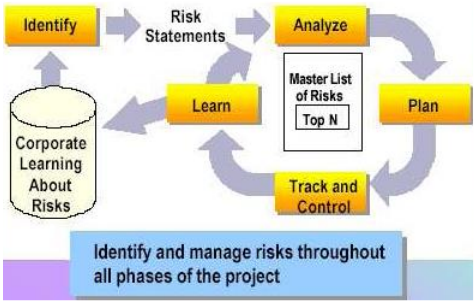
En esta etapa se debe determinar cuándo desistir y finalizar el proyecto. Se debe marcar la relación entre cada factor de riesgo enumerado y el punto de referencia y definir el área de incertidumbre, donde será tan válido continuar como interrumpir el trabajo y predecir cómo la combinación de riesgos afectará a los niveles de referencia.

Proceso de Gestión de Riesgos Proactiva

Identificación de los Riesgos.

- Análisis (Definir probabilidad e impacto).
- Planificación (contingencia, mitigación).
- Monitoreo y Control (Supervisión).
- Aprendizaje.

Este proceso se realiza continuamente, y permite obtener una base de datos que sirva como base de referencia para los próximos proyectos.



Software

El software es un transformador de información, realiza la producción, el manejo, la adquisición, la modificación, el despliegue o la transmisión de la información que puede ser tan simple como un solo bit o tan compleja como una presentación multimedia. El software entrega el producto más importante de nuestro tiempo: la información.

Desde el punto de vista del ingeniero de software, el producto obtenido la forman los programas, el contendido (datos) y los documentos que constituyen el software. Pero desde el enfoque del usuario, el producto obtenido es la información resultante que de alguna manera mejora el mundo del usuario.

Características del software

El software es un elemento lógico, en lugar de físico, de un sistema. Por lo tanto, el software tiene características muy diferentes a las del hardware:

- El software se desarrolla o construye, no se manufactura en el sentido clásico.
- El software no se desgasta, pero si se deteriora: El software es inmune a los males ambientales que desgastan al hardware. La curva de la tasa de fallas para el software debería tener la forma de la

"curva idealizada". Los defectos sin descubrir causan tasas de falla altas en las primeras etapas de vida de un programa. Sin embargo, los errores se corrigen y la curva se aplana. Sin embargo, el software se deteriora: Durante su vida, el software sufre cambios, lo cual puede introducir errores, y ocasiona que la curva de fallas tenga un pico. Antes de que la curva pueda volver a su estado original, se requiere otro cambio lo que ocasiona que la curva tenga otro pico. Por lo tanto, el mantenimiento del software es más complejo que el del hardware.

- A pesar de que la industria tiene una tendencia hacia la construcción por componentes, la mayoría del software aún se construye a medida: Los componentes reutilizables se han creado para que el ingeniero se pueda concentrar en los elementos que en realidad son innovadores en el diseño. Esto es muy utilizado en el diseño de hardware, pero en el ámbito del software recién se esta comenzando a extender.

CONCEPTOS DE GESTIÓN DE PROYECTOS

CONCEPTOS DE GESTION DE PROYECTOS

Actividad protectora dentro de la ingeniería del software. Comienza antes de iniciar cualquier actividad técnica y continúa a lo largo de la definición, el desarrollo y el soporte del software de computadora.
Esta actividad abarca medidas y métricas, estimación y planificación análisis de riesgos, seguimiento y control.
La gestión eficaz de la gestión de proyectos de software se enfoca sobre las cuatro P: personal, producto, proceso y proyecto.

El personal

El factor humano es tan importante que el Software Engineering Institute ha desarrollado un modelo de madurez de la capacidad de gestión del personal (MMCGP).
El modelo de madurez de gestión de personal define las siguientes áreas clave prácticas para el personal de software: reclutamiento, selección, gestión del desempeño, entrenamiento, retribución, desarrollo de la carrera, diseño de la organización y el trabajo, y desarrollo de la cultura del equipo. Las organizaciones que logran altos niveles de madurez en el área de gestión de personal tienen una mayor probabilidad de implementar efectivas prácticas de ingeniería de software.
Debe estar organizado en equipos eficientes, motivados para hacer un trabajo de software de alta calidad y coordinados para lograr una comunicación eficaz.

El producto

Antes de planear un proyecto se deberían establecer los objetivos y el ámbito del producto, considerar soluciones alternativas e identificar las restricciones técnicas y de gestión.
El desarrollador del software y el cliente se deben reunir para definir los objetivos y el ámbito del producto.
Los objetivos identifican las metas globales del producto, sin considerar cómo se las lograrán.
El ámbito identifica los datos primarios, las funciones y los comportamientos que caracterizan al producto y los intentos por enlazar tales características en una forma cuantitativa.
Una vez entendidos los objetivos y el ámbito del producto se consideran soluciones alternativas. Aunque se trata relativamente poco detalle, las alternativas posibilitan que los gestores y practicantes seleccionen un "mejor" enfoque, cúmplanlas restricciones que imponen las fechas límites de entrega, las restricciones presupuestarias, la disponibilidad del personal, las interfaces técnicas y miles de factores más.
Los requisitos del producto se deben comunicar del cliente al desarrollador, ser divididos en sus partes constitutivas y distribuirse para que trabaje el equipo de software.

El proceso

Un proceso de software proporciona el marco de trabajo desde el cual se puede establecer un plan detallado para el desarrollo del software. Un pequeño número de actividades del marco de trabajo es aplicable a todos los proyectos de software, sin importar su tamaño o complejidad. Algunos conjuntos de tareas diferentes (tareas, hitos, productos de trabajo y puntos de control de calidad) permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software, así como a los requisitos del equipo del proyecto. Finalmente, las actividades protectoras (control de calidad del software,

la gestión de configuración del software y la medición) cubren el modelo de proceso. Las actividades protectoras son independientes de cualquier actividad del marco de trabajo y ocurren durante todo el proceso.

Debe adaptarse al personal y al problema. Se selecciona un marco de trabajo de proceso común, se aplica un paradigma de ingeniería de software adecuado y se elige un conju8nto de tareas para llevar a cabo el trabajo.

El proyecto

Los proyectos de software se realizan de manera planificada y controlada por una razón principal: es la única forma conocida de gestionar la complejidad.

Para evitar el fracaso del proyecto, un gestor de proyecto de software y los ingenieros de software que construyen el producto deben eludir un conjunto de señales de advertencia comunes, comprender los factores de éxito críticos que conducen a una buena gestión del proyecto y desarrollar un enfoque de sentido común para planificar, supervisar y controlar el proyecto.

Debe estar organizado en una forma que permita triunfar al equipo de software.

PERSONAL

Los gestores argumentan que las personas son lo principal en los procesos de ingeniería de software, pero sus acciones con frecuencia contradicen sus palabras.

El elemento central de todos los proyectos de software es el PERSONAL. Los ingenieros de software pueden organizarse en diferentes estructuras de equipo que van desde las jerarquías de control tradicionales hasta los equipos de paradigma abierto.

Se pueden aplicar varias técnicas de coordinación y comunicación para apoyar el trabajo del equipo.

Los participantes

El proceso de software lo integran participantes que pueden clasificarse dentro de una de cinco categorías:

- **Gestores ejecutivos:** definen los aspectos del negocio que usualmente tienen una influencia significativa en el proyecto.
- **Gestores del proyecto:** quienes planifican, motivan, organizan y controlan a los profesionales que realizan el trabajo del software.
- **Profesionales:** quienes proporcionan las habilidades técnicas necesarias para realizar la ingeniería de un producto o aplicación.
- **Clientes:** quienes especifican los requisitos para la ingeniería del software y otros elementos que tienen un interés mínimo en el resultado.
- **Usuarios finales:** quienes interactúan con el software una vez que se libera para su uso productivo.

Lideres de equipo

La gestión del proyecto es una actividad intensamente humana, por lo tanto, los profesionales competentes con frecuencia no son buenos líderes de equipo. Simplemente no tienen la mezcla correcta de habilidades con el personal.

Weinberg sugiere un modelo MOI de liderazgo, basado en la **M**otivación, **O**rganización e **I**nnovación.

Otra visión de las características que definen un gestor de proyecto eficiente resalta 4 rasgos clave: Resolución de problemas, Dotes de gestión, Incentivos e Influencia y fomento de la cultura de equipo.

El equipo de software

Existen casi tantas estructuras organizacionales de profesionales para el desarrollo de software como organizaciones que tiene el mismo fin. La estructura organizacional no puede ser fácilmente modificada. Las preocupaciones acerca de las consecuencias prácticas y políticas de cambio organizacional no están dentro del ámbito de responsabilidad del gestor del proyecto de software. Sin embargo, la organización de la gente directamente involucrada en un proyecto de software está dentro del ámbito del gestor del proyecto.

Los factores de proyecto que deberían considerarse cuando se planifica la estructura de los equipos de ingeniería del software son:

- La dificultad del problema que se resolverá.

- El tamaño del programa resultante en líneas de código o puntos de función.
- La vida del equipo.
- El grado en el que el problema puede separarse en módulos.
- La calidad y confiabilidad requeridas del sistema que se construirá.
- La rigidez de la fecha de entrega.
- El grado de sociabilidad que requiere el proyecto.

Constantine, sugiere 4 paradigmas organizacionales para los equipos de ingeniería de software:

- **Paradigma cerrado**, estructura un equipo a lo largo de una jerarquía tradicional de autoridad. Estos equipos pueden trabajar mejor cuando producen software muy similar a los proyectos anteriores, pero será menos probable que sean innovadores.
- **Paradigma aleatorio**, estructura un equipo libremente y depende de la iniciativa individual de los miembros del equipo. Cuando se requieren innovación o adelantos tecnológicos, los equipos que siguen el paradigma aleatorio serán excelentes, pero no son recomendables cuando se requiere desempeño ordenado.
- **Paradigma abierto,** el trabajo se desarrolla en colaboración. La sólida comunicación y la toma de decisiones basada en el consenso son las marcas características de los equipos de paradigma abierto. Las estructuras de equipo de paradigma abierto se adecuan bien a la solución de problemas complejos, pero no pueden desempeñarse de manera tan eficiente como otros equipos.
- **Paradigma sincrónico,** organiza a los miembros del equipo para trabajar en partes del problema con poca comunicación activa entre ellos.

Equipos ágiles

La filosofía ágil alienta la satisfacción del cliente y la temprana entrega incremental de software; pequeños equipos de trabajo enormemente motivados; métodos informales; mínimos productos de trabajo de ingeniería del software y simplicidad global de desarrollo.

El enfoque ágil subraya la competencia individual en conjunción con la colaboración del grupo como factores de éxito cruciales para el equipo.

Para aprovechar en forma eficiente las competencias de cada miembro del equipo y fomentar la colaboración eficaz a lo largo de un proyecto de software, los equipos ágiles son “auto organizados”. Un equipo auto organizado no necesariamente mantiene una sola estructura de equipo, sino que más bien aprovecha elementos de los paradigmas aleatorio, abierto y sincrónico.

Conflictos de coordinación y comunicación

Existen muchas razones por las cuales los proyectos de software se vuelven problemáticos. La escala de muchos esfuerzos de desarrollo es grande, lo que conduce a complejidad, confusión y dificultades significativas en la coordinación de los miembros del equipo.

EL PRODUCTO

Ámbito del software

La primera actividad de gestión de un proyecto de software es la determinación del ámbito de software. El ámbito se define al responder las siguientes preguntas:

- **CONTEXTO** ¿Cómo encaja el software que se desarrollará en un sistema más grande, producto o contexto de negocios, y qué restricciones se imponen como resultado del contexto?
- **OBJETIVOS DE INFORMACION** ¿Qué objetos de datos visibles al usuario se producen como resultado del software? ¿Qué objetos de datos se requieren de entrada?
- **FUNCION Y DESEMPEÑO ¿**Qué funciones realiza el software para transformar los datos de entrada en salida? ¿Existen algunas características de desempeño especiales que deban abordarse?

El ámbito del proyecto de software no debe ser ambiguo ni incomprensible a niveles de gestión y técnico. Se debe acotar un enunciado del ámbito del software, es decir, se establecen de manera explícita los datos cuantitativos, se anotan las restricciones o limitaciones y se describen los factores que reducen riesgos.

Descomposición del problema

La descomposición del problema, a veces llamada partición o elaboración del problema, es una actividad que se asienta en el núcleo del análisis de requisitos de software. La descomposición se aplica en dos grandes áreas:

- La funcionalidad que debe entregarse.
- El proceso que se empleará para entregarlo.

Un problema complejo se divide en problemas menores que resultan más manejables. Ésta es la estrategia que se aplica cuando comienza la planificación del proyecto. Las funciones de software se evalúan y refinan para proporcionar más detalles antes del comienzo de la estimación. Puesto que las estimaciones de costo y planificación temporal están funcionalmente orientadas, con frecuencia es útil cierto grado de descomposición.

EL PROCESO

El problema que se presenta es seleccionar el modelo de proceso apropiado para que un equipo de proyecto someta al software a ingeniería.

El gestor de proyectos debe decidir cuál modelo de proceso es más apropiado para:

- Los clientes que han solicitado el producto y el personal que hará el trabajo.
- Las características del producto mismo.
- El ambiente del proyecto en el que trabaja el equipo de software.

Cuando se ha seleccionado un modelo de procesos entonces el equipo define un plan de proyecto preliminar con base en el conjunto de actividades del marco del trabajo del proceso.

Una vez que se establece el plan preliminar, comienza le descomposición del proceso.

Combinación del producto y el proceso

La planeación del proyecto comienza con la combinación del producto y del proceso. Cada función que el equipo de software someterá a ingeniería debe pasar a través del conjunto de actividades del marco de trabajo definidas para una organización de software.

Descomposición del proceso

Un equipo de software debe tener un grado significativo de flexibilidad al elegir el modelo de procesos de software que sea mejor para el proyecto y las tareas de ingeniería de software que integren el modelo de procesos una vez elegido.

- **Enfoque secuencial lineal:** para realizar un proyecto relativamente pequeño similar a otros que se hayan realizado.
- **Modelo de desarrollo rápido de aplicaciones:** utilizado cuando existen restricciones de tiempo muy ceñidas.
- **Estrategia Incremental:** utilizado cuando la fecha límite es tan ceñida que la funcionalidad completa no puede alcanzarse.

Proyectos con otras características conducirán a la búsqueda de otros modelos. Una vez elegido el modelo de proceso, el marco de trabajo respectivo se adapta a él. Podrá aplicarse el marco de trabajo genérico: comunicación, planificación, modelado, construcción y despliegue. Funcionará para modelos lineales, iterativos e incrementales, así como evolutivos e incluso para modelos concurrentes o de ensamble de componentes.

La descomposición del proceso comienza cuando el gerente de proyecto pregunta ¿Cómo lograremos esta actividad del marco de trabajo?

EL PROYECTO

La gestión de un proyecto de software exitoso requiere entender que puede salir mal. John Reel define10 señales que indican que un proyecto de sistemas de información está en peligro:

1. El personal de software no entiende las necesidades de sus clientes.
2. El ámbito del producto está mal definido.
3. Los cambios se gestionan mal.

4. La tecnología elegida cambia.
5. Las necesidades comerciales cambian, o están mal definidas.
6. Los plazos de entrega no son realistas.
7. Los usuarios se resisten.
8. Se pierde el patrocino.
9. El equipo de proyecto carece de personal.
10. Los gestores evitan las mejores prácticas y las lecciones aprendidas.

La lista de las señales mencionada, son las causas que conducen a la regla 90-90. El primer 90% de un sistema absorbe el 90% del esfuerzo y tiempo asignados. El último 10% toma el otro 90% del esfuerzo y tiempos asignados.

¿Cómo actúa un gestor para evitar los problemas mencionados? Sugiere un enfoque de sentido común de 5 partes para proyectos de software:

1. **Comience con el pie derecho:** Entender bien el problema para establecer bien los objetivos y expectativas. Construir el equipo correcto y darle a éste autonomía, autoridad y tecnología.
2. **Mantenga el ímpetu:** El gestor de proyecto debe proporcionar incentivos, el equipo debe resaltar la calidad en cada tarea que realiza y los gestores ejecutivos deben hacer lo posible por mantenerse fuera del camino del equipo.
3. **Rastree el progreso:** En un proyecto de software el progreso se rastrea conforme se elaboran los productos de trabajo(código fuente-modelos) y se aprueban como parte de una actividad de aseguramiento de calidad
4. **Tomar decisiones inteligentes**: Las decisiones del gestor de proyecto y del equipo de software deben encaminarse para mantenerlo simple.
5. **Realice un análisis de resultados:** Establezca un mecanismo consistente para extraer lecciones aprendidas por cada proyecto. Evalúe la planificación real y la prevista, recolecte y analice métricas, obtenga realimentación por parte del equipo y de los clientes.

ESTIMACIÓN PARA PROYECTOS DE SOFTWARE

Gestión del proyecto del software: comienza con un conjunto de actividades que en grupo se llaman PLANIFICACION DEL PROYECTO. 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.
Una vez que se completen estas actividades, el equipo de software debe establecer un Plan De Proyecto, que defina las tareas y fechas clave de la ingeniería del software, que identifique quién es el responsable de dirigir cada tarea y especifique las dependencias entre tareas que pueden ser determinantes del progreso.

OBSERVACIONES ACERCA DE LA ESTIMACION

La planificación requiere que los gestores técnicos y los miembros del equipo de software establezcan un compromiso inicial. Aunque la estimación es tanto un arte como una ciencia, esta importante actividad no necesita realizarse en una forma improvisada.
Puesto que la estimación coloca los cimientos para las demás actividades de planificación del proyecto y ésta proporciona la ruta para la ingeniería del software exitosa, se estaría mal aconsejando si se embarcara sin ella.
El riesgo de estimación se mide por el grado de incertidumbre de las estimaciones Cuantitativas establecidas para recursos, costos y programa de trabajo

EL PROCESO DE PLANIFICACION DEL PROYECTO

El objetivo de la planificación del proyecto de software es proporcionar un marco de trabajo que permita al gestor estimar razonablemente recursos, costo y programa de trabajo. Además, las estimaciones deben intentar definir los escenarios de mejor y peor caso de modo que los resultados del proyecto se puedan acotar.

El plan del proyecto se debe adaptar y actualizar conforme avance el proyecto.

AMBITO DEL SOFTWARE Y FACTIBILIDAD

El ámbito del software describe las funciones y características que se entregarán a los usuarios finales, los datos que son entrada y salida, el contenido que se presenta a los usuarios como consecuencia de emplear el software, así como el desempeño, las restricciones, las interfaces y la confiabilidad que acotan al sistema. El ámbito se define al usar una de las dos técnicas siguientes:

- Después de una comunicación con todos los participantes se desarrolla una descripción narrativa del ámbito del software.
- Los usuarios finales desarrollan un conjunto de casos de uso.

Una vez identificado el AMBITO es razonable preguntar ¿Es posible construir el software para satisfacer el ámbito? ¿El proyecto es factible?
El AMBITO no es suficiente para contestar las preguntas mencionadas. Una vez que el ámbito se comprende el equipo de software y otros deben trabajar para determinar si se puede hacer dentro de las siguientes dimensiones:

- Tecnología
- Finanzas
- Tiempo
- Recursos

RECURSOS

La segunda tarea de la planificación es la estimación de los recursos necesarios para completar el esfuerzo de desarrollo de software.

Recursos humanos

El planificador comienza evaluando el ámbito del software y seleccionando las habilidades requeridas para completar el desarrollo. Se especifican tanto la posición organizacional como la especialidad.
El número de personas que requiere un proyecto de software solo se determina después de que se ha hecho una estimación del esfuerzo de desarrollo.

Recursos de software reutilizables

La ingeniería del software basada en componentes enfatiza la reutilización, es decir, la creación y reutilización de bloques de construcción de software. Tales bloques, usualmente llamados componentes, deben catalogarse para consultarlos con facilidad, estandarizarse para facilitar su aplicación y validarse para integrarlos fácilmente.
Bennatan sugiere 4 categorías:

- **Componentes ya desarrollados:** El software existente se puede adquirir de un tercero o se desarrolló internamente para un proyecto previo (están listos y validados).
- **Componentes experimentales:** Especificaciones, diseño, código o datos de prueba existentes que se desarrollaron para proyectos previos similares.
- **Componentes de experiencia parcial:** Especificaciones, diseño, código o datos de prueba existentes que se desarrollaron para proyectos previos están relacionados con el proyecto actual, pero requerirán modificaciones sustanciales.
- **Componentes nuevos:** El equipo de software debe construir los componentes de software específicamente para las necesidades del proyecto actual.

Recursos del entorno

El entorno que soporta un proyecto de software, con frecuencia denominado entorno de ingeniería de software (EIS), incorpora hardware y software. El hardware proporciona una plataforma que soporta herramientas (software) con que se producen los productos de trabajo basados en una buena práctica de la ingeniería del software.
El planificador del proyecto de software debe especificar cada elemento de hardware.

ESTIMACION DE PROYECTOS DE SOFTWARE

La estimación de costo y esfuerzo nunca será una ciencia exacta. Sin embargo, la estimación del proyecto se puede transformar de una práctica oscura en una serie de pasos sistemáticos que proporcionan estimaciones con riesgo aceptable.

Para lograr estimaciones confiables de costo y esfuerzo se tienen varias opciones:

- Demorar la estimación hasta más tarde en el proyecto. (Poco práctica, las estimaciones deben realizarse por adelantado).
- Basar las estimaciones en proyectos similares que ya hayan sido completados (Puede funcionar si el proyecto en curso es muy similar a los previos).
- Emplear técnicas de descomposición relativamente simples para generar estimaciones de costo y esfuerzo del proyecto.
- Utilizar uno o más modelos empíricos en la estimación de costo esfuerzo.

Las últimas 2 opciones son enfoques viables, deben aplicarse juntas, cada una empleada como una marca de verificación para la otra.

TECNICAS DE DESCOMPOSICION

La estimación del proyecto de software es una forma de resolver problemas, en la mayoría de los casos complejos como para considerar una sola pieza. Por ello, se descompone el problema.

Tamaño del software

La precisión de la estimación de un proyecto de software se manifiesta en varios factores:

1. El grado con el cual el planificador ha estimado adecuadamente el tamaño del producto que se construirá.
2. La habilidad para traducir la estimación del tamaño en esfuerzo humano, programa de trabajo y dinero.
3. El grado en el cual el plan del proyecto refleja las habilidades del equipo de software.
4. La estabilidad de los requisitos del producto y el entorno que soporta el esfuerzo de ingeniaría de software.

Putnam y Myers sugieren 4 enfoques al problema del tamaño:

1. **Tamaño de lógica difusa:** El planificador identifica el tipo de aplicación, establece su magnitud.
2. **Tamaño de punto de función:** El planificador desarrolla estimaciones de las características del dominio de la información**.**
3. **Tamaño de componentes estándar:** El planificador del proyecto estima el número de ocurrencias de cada componente estándar y luego aplica datos de proyectos históricos para determinar el tamaño de entrega por componente estándar.
4. **Tamaño del cambio:** El planificador estima el número y tipo de las modificaciones que se deben lograr

Estimación basada en el problema

Las estimaciones de LDC y PF son distintas técnicas de estimación, aunque ambas tienen varias características en común.

Las técnicas de estimación LDC y PF difieren en cuanto al detalle requerido para descomposición y el objetivo de la partición.

Mientras mayo sea el grado de partición es más probable que se desarrolle una estimación razonablemente precisa de LDC.

Estimación basada en el proceso

La técnica más común para estimar un proyecto es basar la estimación en el proceso que se empleará. Es decir, el proceso se descompone en un conjunto relativamente pequeño de tareas y se estima el esfuerzo requerido para lograr cada tarea.

La estimación basada en el proceso comienza con un bosquejo de las funciones del software obtenidas a partir del ámbito del proyecto. Cada función requiere realizar un grupo de actividades del marco de trabajo.

Una vez que se combinan las funciones del problema y las actividades del proceso, el planificador estima el esfuerzo que se requerirá para lograr cada actividad del proceso de software en cada función. Luego se aplica la tasa de trabajo promedio (costo/unidad de esfuerzo) al esfuerzo estimado para cada actividad del proceso.

Estimación con casos de uso

Los casos de uso permiten que un equipo de software comprenda el ámbito del software y los requisitos. Razones por las cuales, desarrollar un enfoque de estimación con casos de uso, es problemático:

- Los casos de uso se describen empleando muchos formatos y estilos diferentes, no existe un formato estándar.
- Los casos de uso representan una visión externa de visión externa (del usuario), del software y con frecuencia están escritos con diferentes grados de abstracción.
- Los casos de uso no abordan la complejidad de las funciones ni de las características que se describen.
- Los casos de uso no describen el comportamiento complejo que involucran muchas funciones y características.

A diferencia de las LDC o PF, el caso de uso de una persona tal vez requiera meses de esfuerzo mientras que el de otra quizá se implemente en un día o dos.

MODELOS EMPIRICOS DE ESTIMACION

Los datos empíricos que apoyan la mayoría de los modelos de estimación proceden de una manera limitada de proyectos. Por esta razón ningún modelo de estimación es apropiado para todas las clase de software ni en todos los entornos de desarrollo.

Un modelo de estimación debe calibrarse para reflejar las condiciones locales. El modelo debe probarse mediante la aplicación de los datos recopilados a partir de proyectos completados, colocar los datos en el modelo y luego comparar los resultados reales con los predichos.

ESTIMACION PARA PROYECTOS ORIENTADOS A OBJETOS

Enfoque planteado explícitamente para software OO:

1. Desarrollar estimaciones aplicando la descomposición del esfuerzo, análisis de PF y cualquier otro método que sea aplicable en aplicaciones convencionales.
2. Aplicar el modelado de análisis OO.
3. Determinar el número de clases clave.
4. Categorizar el tipo de interfaz para la aplicación.
5. Multiplicar el número total de clases por el número promedio de unidades de trabajo por clase.
6. Comprobar de manera cruzada la estimación.

CALENDARIZACIÓN DE PROYECTOS DE SOFTWARE

Aunque existen muchas razones por las cuales el software se entrega con retraso, la mayoría se encuadra en una o más de las siguientes causas:

- Una fecha límite irrealizable establecida por alguien externo al grupo de ingeniería del software e impuesta a los gestores y profesionales del grupo.
- Cambios en los requisitos del cliente que no se reflejan en modificaciones a la calendarización.
- Una subestimación razonable de la cantidad de esfuerzo o de recursos que se requerirán para alcanzar el trabajo.
- Riesgos que no se consideraron cuando empezó el proyecto.
- Dificultades técnicas que no pudieron preverse.
- Dificultades humanas imprevisibles.
- Falta de comunicación entre el personal del proyecto, lo que genera demoras.
- Una falla en la gestión del proyecto porque no reconoció el retraso ni emprendió una acción para corregir el problema.

Pasos a seguir ante una situación de fecha límite:

1. Realizar una estimación detallada usando datos de proyectos previos. Determinar esfuerzo y duración estimados.
2. Aplicar un modelo de proceso incremental. Desarrollar una estrategia de ingeniería de software que entregará la funcionalidad crítica en la fecha límite impuesta, pero demorará otra.
3. Reunirse con el cliente y con la estimación detallada, explicarle por qué la fecha límite impuesta es irrealizable. Asegurarse de señalar que todas las estimaciones están basadas sobre el desempeño en proyectos previos.
4. Ofrezca la estrategia de desarrollo incremental como alternativa.

CALENDARIZACION DE PROYECTO

Cómo se retrasan los proyectos de software en la calendarización y contestó: DE UN DIA A LA VEZ
La realidad de un proyecto técnico es que cientos de pequeñas tareas deben realizarse para lograr una meta mayor.
Si las tareas con "trayectoria crítica" se retrasan en la calendarización, la fecha de terminación del proyecto se pone en riesgo.

Objetivo del Gestor: Definir todas las tareas del proyecto, construir una red que bosqueje sus interdependencias, identificar las tareas cruciales dentro de la red y luego seguir su progreso para garantizar que la demora se produce "un día a la vez".

Calendarización del proyecto de software: es una actividad que distribuye estimaciones de esfuerzo a través de la duración planificada del proyecto al asignar el esfuerzo a tareas específicas de ingeniería de software. La calendarización evoluciona a lo largo del tiempo.

La calendarización para proyectos de ingeniería de software se puede ver desde 2 perspectivas:

- Ya se ha establecido una fecha final para la liberación de un sistema basado en computadora.
- Supone que se han comentado límites cronológicos aproximados, pero que a la fecha final la establece la organización de ingeniería del software.

PRINCIPIOS BASICOS

- **Compartimentación:** El proyecto debe dividirse en compartimentos en varias actividades, acciones y tareas manejables.(Descomponer tanto el producto como el proceso).
- **Interdependencia:** Se debe determinar la interdependencia de cada actividad, acción o tarea compartimentada.
- **Asignación de tiempo:** A cada tarea por calendarizar se le debe asignar cierto número de unidades de trabajo, fecha de inicio, fecha de terminación.
- **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 más que el número de personas calendarizado.
- **Definición de responsabilidades:** Toda tarea calendarizada se la debe asignar a un miembro específico del equipo.
- **Definición de resultados:** Toda tarea calendarizada debe tener un resultado definido. En proyectos de software generalmente es un producto de trabajo.
- **Definición de hitos:** Un hito se logra cuando se ha revisado la calidad de uno o más productos de trabajo y se ha aprobado

RELACION ENTRE EL PERSONAL Y EL ESFUERZO

MITO: "Si nos retrasamos en la calendarización, siempre podemos incorporar más programadores y recuperarnos más adelante en el proyecto"

Desgraciadamente, agregar más personas en etapas tardías tiene un efecto perturbador sobre éste, lo que provoca que la calendarización se desfase aún más.
Las personas que se agregan tienen que aprender el sistema, y la gente que les enseña es la misma que estaba haciendo el trabajo. Durante la enseñanza no se realiza trabajo y el proyecto experimenta mayores retrasos.

DEFINICION DE UN CONJUNTO DE TAREAS PARA EL PROYECTO DE SOFTWARE

Es difícil desarrollar una taxonomía completa de tipos de proyectos de software, en la mayoría de las organizaciones del ramo se encuentran los siguientes proyectos:

1. Proyectos de desarrollo del concepto, los cuales se inician para explorar algunas aplicaciones o conceptos de negocio s de alguna nueva tecnología.
2. Proyectos de desarrollo de nuevas aplicaciones, los cuales se llevan a cabo como consecuencia de una solicitud específica del cliente.
3. Proyectos de mejora de la aplicación, éstos ocurren cuando el software existente experimenta grandes modificaciones en la función, el desempeño o las interfaces visibles para el usuario final.
4. Proyectos de mantenimiento de aplicación, los cuales corrigen, adaptan o extienden el software existente en formas que no sean obvias inmediatamente para el usuario final.
5. Proyectos de reingeniería, éstos se llevan a cabo con la finalidad de reconstruir un sistema existente en todo o en parte.

CALENDARIZACION

Técnicas que se aplican para la calendarización: PERT, CMP, GANTT.

Seguimiento de la calendarización

La calendarización del proyecto proporciona un mapa de carreteras al gestor del proyecto de software. Si se ha realizado de manera adecuada, la calendarización del proyecto define las tareas e hitos que se deben seguir y controlar conforme avance el proyecto. El seguimiento se puede hacer de diferentes maneras:

- Con la realización periódica de reuniones para valorar el estado del proyecto en las cuales cada uno de los miembros del equipo informa del progreso y los problemas.
- Con la evaluación de los resultados de todas las revisiones realizadas a lo largo del proceso de ingeniería del software.
- Con la determinación de si se han logrado los hitos formales del proyecto en la fecha programada.
- Al comparar la fecha de inicio real con la fecha prevista para cada tarea del proyecto.
- Al reunirse de manera informal con los trabajadores para obtener su evaluación subjetiva del progreso, hasta la fecha y los problemas que se vislumbran.
- Con el uso del análisis del valor obtenido para evaluar el progreso cuantitativamente.

Problemática de la estimación

- Averiguar lo que costará desarrollar una aplicación (personas, dinero, tiempo).
- Momento en que se desea conocer el costo.
- Siempre se quiere muy pronto.

Cada vez que se avanza más en el proyecto, y se obtiene más información, la estimación es más precisa. El problema es realizar predicciones.

Proceso de estimación propuesto

- Medir lo que el usuario quiere: Especificar y cuantificar lo que el usuario necesita.
- Estimar lo que costará:
  - Experiencia individual
  - Experiencia de empresa

Debe tenerse en cuenta la experiencia en proyectos previos. Luego de que se hayan desarrollado varios proyectos, se contará con características claves de negocio.

MÉTODOS UTILIZADOS

- Basados en la experiencia
  - Juicio experto: Puro, Delphi.
  - Analogía
  - Distribución de la utilización de recursos en el ciclo de vida
- Basados exclusivamente en los recursos
- Método basado exclusivamente en el mercado
- Basados en los componentes del producto o en el proceso de desarrollo
- Métodos algorítmicos

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.
- Si el experto no está, el equipo no sabe cómo realizar las estimaciones, se debe tratar de transmitir el conocimiento a todo el equipo.

Estructurando el juicio experto

- Hacer una WBS con una granularidad aceptable
- Usar el método de "Optimista, pesimista y habitual" y su fórmula (0+4h+p)/6.
- Use un checklist y un criterio definido para asegurar cobertura.

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.
- La reunión se prepara 24hs antes, se junta un grupo de expertos para enfrentar opiniones y estimar. Cada uno de los expertos envía su estimación al coordinador. Luego se tabulan las estimaciones (son anónimas).
- Se dan 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 se la envía al coordinador.
- Se repite el proceso hasta que la estimación converge de forma razonable.

¿Cuántas rondas son aceptables? Generalmente 3 reuniones son aceptables.

Analogía

- Consiste en comparar las especificaciones de un proyecto con las de otros proyectos.
- Comparar componentes similares
- Cuando se compara por analogía hay mucho error, no son fiables.

Analogía-Factores

- Tamaño: ¿Mayor o menor?
- Complejidad: ¿Más complejo de lo usual?
- Usuarios: Si hay más usuarios habrán más complicaciones
- Otros factores:
  - Sistema operativo, entornos (la primera vez más).
  - Hardware, ¿Es la primera vez que se va a utilizar?
  - Personal del proyecto, ¿nuevos en la organización?

Generalmente lo que más afecta es la complejidad y para quién se estima.

Estrategia de estimación

- Cuente primero
- Use un método para convertir las "cuentas" en estimaciones
- Use el juicio de experto como último resorte.

> Primero se debe contar y fijar la medida.
> Segundo se debe transformar la cuenta en estimación.
> NO USAR UN SOLO MÉTODO SE RECOMIENDA USAR POR LO MENOS 2, Y SE SUELEN USAR DE A PARES.
> Generalmente se usa el juicio experto y uno algorítmico.

Calibraciones y Historical Data

- Las calibraciones son solamente usadas para convertir las "cuentas" a estimados (líneas de código a esfuerzo, use cases a calendarios, requerimientos a números de test cases, etc.).
- Estimar, siempre involucra algún tipo de calibración sea directa o implícita.
- Las estimaciones pueden ser calibración usando cualquiera de estos tres tipos de datos:
    1. **Industry Data:** datos de otras organizaciones que aportan al mercado y desarrollan productos con algún grado de semejanza y que permite una comparación básica. (En general están es estándares)
    2. **Historical Data:** datos de la organización de proyectos que se desarrollaron y ya se cerraron.(Se guardaron datos-estimaciones)
    3. **Project Data:** datos del proyecto pero de etapas anteriores a la que se está estimando.(Cada vez que se abre una fase estimarla)

Actividades Omitidas

- Una de las fuentes de error más común en las estimaciones es omitir actividades necesarias para la estimación del proyecto:
    - Requerimientos faltantes
    - Actividades de desarrollo faltantes (documentación técnica, participación en revisiones, creación de datos para el testing, mantenimiento de producto en previas versiones).
    - Actividades generales (días por enfermedad, licencia, cursos, reuniones de la compañía)
- Uso de buffers o colchón.
- "Nunca tenga temor de que las estimaciones creadas por desarrolladores sean demasiado pesimistas, dado que los desarrolladores siempre generan schedules demasiado optimistas".

Lo que más afecta a la estimación es:

1. Requerimientos no relevantes
2. Requerimientos mal relevados

Siempre guardar una cantidad de días extra por algo que salga mal o falte: buffer o colchón.

> Factores que afectan a la estimación: Nos olvidamos de los requerimientos NO FUNCIONALES, sólo tenemos en cuenta los funcionales.

ESTIMACIÓN DE TAMAÑO

- El número que más se busca en las estimaciones de software es el tamaño del software a ser construido.
- El tamaño puede ser expresado en LDC (líneas de código), puntos de función, número de requerimientos, número de web pages u otra medida.

Pocker Estimation

- Popular entre los ágiles practitiones, publicado por Mike Cohn.

- Combina opinión de experto, analogía y desegregación.
- Participantes en planning pocker son desarrolladores
    - Las personas más competentes en resolver una tarea deben ser quienes la estiman.
- El misterio de Fibonacci.

Pocker Planning

- Defina la lista de actividades, módulos, use cases, etc.
- Acuerde y consensue en la más simple, ejemplo tarea z.
- Asigne tamaño/complejidad 1 a la tarea z
- Ejecute un wide Band delphi para estimar complejidad/tamaño del resto de la lista, comparado con la más simple y solo asignando valores de COMPLEJIDAD de la serie de Fibonacci. Se puede hacer como una ronda de cartas.
- Estime el esfuerzo requerido para llevar a cabo la tarea z y aplique el multiplicador a todas las actividades.

Estimación basada en casos de uso

- Basada en un conjunto de casos de uso y su complejidad asociada.
- Se agregan valores que modifican el tamaño.
- Se agregan valores que modifican el esfuerzo.

- **Estimación de tamaño:**
    - Complejidad.
    - Actores.
    - Optimización.
    - Reusabilidad.
    - Tecnología.
    - Salidas.
    - Dominio.
- Criterios:
    - Generales.
    - Particulares.
- **Estimación de esfuerzo:**
    - Madurez de la organización.
    - Experiencia en la tecnología.
    - Conocimiento en el proceso.
- Esfuerzo distribuido en el ciclo de vida:
    - Ajuste por solapamiento.
- Criterios
    - Generales.
    - Particulares**.**

Problemas en las estimaciones

- La estimación de tamaño es el paso intelectual más difícil (pero no imposible), y muchas veces se saltea y se trabaja directamente en la estimación del cronograma (tiempo).
- Los clientes y desarrolladores a menudo no reconocen que el desarrollo de software es un proceso gradual, que requiere el refinamiento de las estimaciones tempranas.
- Las organizaciones a menudo no registran, guardan y analizan sus datos históricos de performance de desarrollo de proyecto.
- A menudo es difícil generar cronogramas realistas aceptados por clientes y gerentes.

Proyectos de mantenimiento

- Cuando se está estimando el tamaño para un proyecto de mantenimiento se debe tener en cuenta que insertar una nueva funcionalidad será factible si la arquitectura existente del producto puede acomodarse a dicha función. Caso contrario, el esfuerzo de mantenimiento incrementará el retrabajo de la arquitectura.
- Puede existir una sobre estimación si se utilizan modelos de estimación calibrados para nuevos proyectos.
- A menudo los proyectos de mantenimiento tienen fecha de entrega fija así como un número de personal asignado. Es otro elemento con el que hay que lidiar en el momento de hacer la estimación.

No aplican los métodos de estimación para proyectos nuevos en proyectos de mantenimiento.

Estimaciones en Proyectos pequeños

- Proceso pequeño: trabajan una o dos personas en un tiempo menor a 6 meses.
- Los datos industriales de modelos de estimación que existen no están calibrados para este tipo de proyecto, de modo que son inadecuados y lo mejor es utilizar los datos históricos de la organización.
- Las estimaciones de los proyectos pequeños dependen en gran medida de las capacidades de los individuos que hacen el trabajo. El Personal Software Process es de gran ayuda para estos proyectos.

MÉTRICAS DEL PRODUCTO PARA EL SOFTWARE

Las medidas se emplean para comprender mejor los atributos de los modelos que se crean y para evaluar la calidad de los productos de la ingeniería o de los sistemas que los construyen. Las métricas de producto se recopilan a medida que se realizan las tareas técnicas y proporcionan al ingeniero de software una indicación en **tiempo real** de la eficacia de los modelos de análisis, diseño y código, pero también aportan indicativos de la efectividad de los casos de prueba y la calidad general del software que se construirá. Es importante el hecho de que provean información en tiempo real, ya que permiten descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos.

A veces el esfuerzo por desarrollar medidas precisas de la calidad del software se ve frustrado por la naturaleza subjetiva de la actividad. En todos los casos las métricas representan medidas indirectas, es decir, nunca se mide realmente la calidad, sino alguna manifestación de ésta.

Medidas, métricas e indicadores

- **Medida:** Proporciona una indicación cuantitativa de la extensión, la cantidad, la dimensión, la capacidad o el tamaño de algún atributo de un producto o proceso.
- **Medición:** Es el acto de determinar una medida.
- **Métrica (Según el IEEE):** Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado.
- **Indicador:** Es una métrica o una combinación de métricas que proporciona conocimientos que permiten al líder ajustar el proceso, el proyecto o el producto para que las cosas mejoran.

Una métrica de software relaciona de alguna manera las medidas individuales.

Principios y Directrices de métricas

- Siempre que sea posible, deben automatizarse la recopilación de datos y su análisis.
- Debe aplicarse técnicas estadísticas válidas para establecer relaciones entre los atributos internos del producto y las características de calidad externas.
- **Cada métrica debe validarse empíricamente en una amplia variedad de contextos antes de publicarse o aplicarse a la toma de decisiones:** Una métrica debe "crecer" para aplicarse en sistemas grandes y funcionar en diversos lenguajes de programación y dominios de sistemas.

Aunque la formulación, caracterización y validación son críticas, la recopilación y el análisis son actividades que dirigen el proceso de medición.

- Para cada métrica deben establecerse recomendaciones para la interpretación.

Paradigma OPM

Es una técnica para identificar métricas significativas aplicables en cualquier parte del proceso de software. El OPM (Objetivo – Pregunta – Métrica) destaca la necesidad de:

- Establecer un **objetivo** de medición explícito que sea específico para la actividad del proceso o las características del producto que se está evaluando.
- Definir un conjunto de **preguntas** que deben responderse con el fin de alcanzar el objetivo.
- Identificar **métricas** bien formuladas que ayuden a responder esas preguntas.

Atributos de las métricas efectivas del software

- **Simples y calculables:** Debe ser relativamente fácil aprender a derivar la métrica y su cálculo no debe consumir demasiado tiempo o esfuerzo. **Una métrica debe tener propiedades matemáticas deseables:** Es decir, el valor de la métrica debe estar en un rango significativo (de 0 a 1 por ejemplo), y realizar la medición en la escala que se mide el componente.
- **Empírica e intuitivamente persuasivas:** La métrica debe satisfacer las nociones intuitivas del ingeniero acerca del atributo del producto que se está construyendo.
- **Consistentes y objetivas:** La métrica siempre debe arrojar resultados que no permitan ninguna ambigüedad.
- **Consistentes en el uso de unidades y dimensiones:** El cálculo debe emplear medidas que no lleven a combinaciones extrañas de unidades. **Cuando se representa una característica de software que aumenta cuando se presentan rasgos positivos o que disminuye al encontrar rasgos indeseables, el valor de la métrica debe aumentar o disminuir en el mismo sentido.**
- **Independientes del lenguaje de programación:** Las métricas deben basarse en le modelo de análisis o diseño, o en la estructura del programa.
- **Proporcionar retroalimentación efectiva:** Es decir, la métrica debe llevar a un producto final de la más alta calidad.

Tipos de métricas

- **Métricas para el modelo de análisis:** Examinan el modelo de análisis con la intención de predecir el "tamaño" del sistema resultante, lo cual a veces sirve como indicador de la complejidad del diseño y del esfuerzo de codificación, integración y prueba.
  Estas métricas incluyen:
  - **Funcionalidad entregada:** Proporciona una medida indirecta de la funcionalidad que se empaqueta con el software.
  - **Tamaño del sistema:** Mide el tamaño general del sistema, definido desde el punto de vista de la información disponible como parte del modelo de análisis.
  - **Calidad de la especificación:** Proporciona una indicación del grado en que se ha completado la especificación de los requerimientos.

- **Métricas para el modelo de diseño:** Cuantifican los atributos del diseño de manera tal que le permiten al ingeniero de software evaluar la calidad del diseño. La métrica incluye:

  - **Métricas arquitectónicas:** Proporcionan un indicio de la calidad del diseño arquitectónico, teniendo en cuenta la estructura arquitectónica y la efectividad de módulos o componentes dentro de la arquitectura.
  - **Métricas al nivel de componente:** Se concentran en las características internas de un componente de software e incluyen medidas de cohesión, acoplamiento y complejidad del módulo.

- **Métricas de diseño de la interfaz:** Se concentran en la calidad, la cohesión y la facilidad de uso.
- **Métricas especializadas en diseño orientado a objetos:** Miden características de clases además de las correspondientes a comunicación y colaboración.

- **Métricas para el código fuente:** Miden el código fuente y se usan para evaluar su complejidad, además de la facilidad con que se mantiene y prueba.

    - **Métricas de Halstead:** Son muy buenas aunque son controversiales. Proporcionan medidas únicas de un programa de cómputo.
    - **Métricas de complejidad:** Miden la complejidad lógica del código fuente.
    - **Métricas de longitud:** Proporcionan un indicio del tamaño del software.

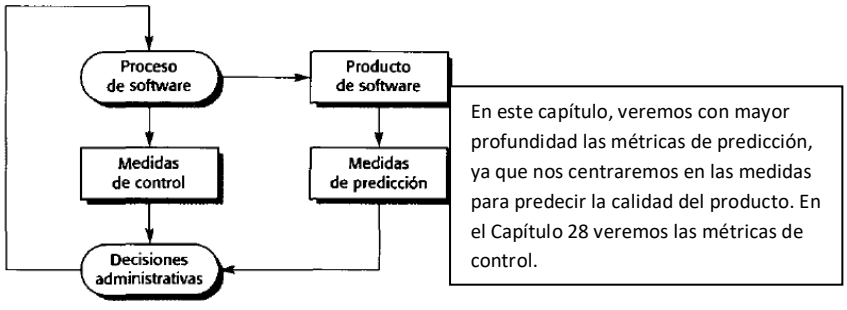
- **Métricas para pruebas:** Casi todas estas métricas se centran en el proceso de prueba, no en las características técnicas de las mismas. En general quienes aplican las pruebas deben depender de las métricas de análisis, diseño y código como guía para el diseño y la ejecución de casos de prueba. Incluyen:

    - **Métricas de cobertura de instrucciones y ramas:** Lleva al diseño de casos de prueba que proporcionan cobertura del programa.
    - **Métricas relacionadas con los defectos:** Se concentran en encontrar defectos y no en las propias pruebas.
    - **Efectividad de la prueba:** Proporcionan un indicio en tiempo real de la efectividad de las pruebas aplicadas.
    - **Métricas en el proceso:** Son métricas relacionadas con el proceso que se determinan a medida que se aplican las pruebas.

27.5 Medición y métricas del software

Sería posible acelerar el proceso de revisión utilizando herramientas que procesaran el diseño del software o el programa, e hiciesen valoraciones automáticas de la calidad del software. Estas valoraciones permiten comprobar que el software tiene el umbral de calidad requerido, y destacar las partes en las cuales no se ha alcanzado para revisarlas.
La **medición del software** se refiere a derivar un valor numérico desde algún atributo del software o del proceso software. Comparando estos valores entre sí y con los estándares aplicados en la organización, es posible sacar conclusiones de la calidad del software o de los procesos para desarrollarlo.
Las **mediciones del software pueden utilizarse para**:
1. Hacer predicciones generales acerca del sistema.
2. Identificar componentes anómalos.

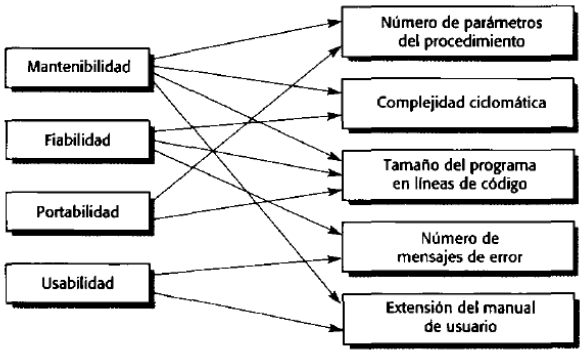
Las métricas son de control o de predicción. Las **métricas de control** suelen estar asociadas con los **procesos**, mientras que las **métricas de predicción** lo están a los **productos**. Ejemplos de las métricas de control o de procesos son el esfuerzo y el tiempo promedio requeridos para reparar los defectos encontrados. Ejemplos de métricas de predicción son la complejidad ciclomática de un módulo, la longitud media de los identificadores de un programa, y el número de atributos y operaciones asociadas con los objetos de un diseño.
Ambas pueden influir en la toma de decisiones de gestión como muestra la Figura siguiente.



Métricas de predicción y de control

Frecuentemente, es imposible medir los atributos de calidad del software directamente. Los atributos de calidad como la mantenibilidad, la comprensión y la usabilidad son atributos externos que nos dicen cómo ven el software los

desarrolladores y los usuarios. Éstos se ven afectados por diversos factores y no existe un camino simple para medirlos. Más bien es necesario medir atributos internos del software (como su tamaño) y suponer que existe una relación entre lo que queremos medir y lo que queremos saber.
La siguiente figura muestra algunos atributos externos de la calidad que nos serán interesantes y los atributos internos relativos a ellos.

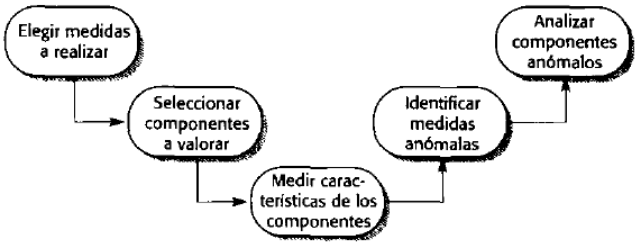


Relaciones entre los atributos internos y externos del software

Para que la medida del atributo interno sea un indicador útil de la característica externa, se deben cumplir tres **condiciones**:
1. El atributo interno debe medirse de forma precisa.
2. Debe existir una relación entre lo que se puede medir y el atributo de comportamiento externo.
3. Esta relación se comprende, ha sido validada y se puede expresar en términos de una fórmula o modelo.

27.5.1 El proceso de medición

Representación del proceso de medición del software dentro de un proceso de control de calidad:



Cada uno de los componentes del sistema se analiza por separado y los diversos valores de las métricas se comparan entre sí y, quizás, con los datos históricos de medición recogidos en los proyectos previos. Las medidas anómalas se utilizan para centrar el esfuerzo de garantía de calidad en los componentes que tienen problemas de calidad.

Los pasos clave en este proceso son:

1. Seleccionar las medidas a realizar. Se deben formular las preguntas que la medición intenta responder y definir las mediciones requeridas para resolver estas preguntas. **Formulación:** La derivación de medidas y métricas apropiadas para la representación del software que se considera.
2. Seleccionar los componentes a evaluar. Se elige un conjunto representativo de componentes.
3. Medir las características de los componentes. Se miden los componentes seleccionados y se calculan los valores de las métricas. **Recolección:** El mecanismo o forma con el que se acumulan los datos necesarios para derivar las métricas formuladas.
4. Identificar las mediciones anómalas. Una vez que se obtienen las mediciones de los componentes, se comparan entre sí y con las mediciones previas registradas en una base de datos de mediciones.
5. Analizar los componentes anómalos. Una vez identificados los componentes con valores anómalos para, las métricas particulares, se examinan estos componentes para decidir si los valores de la métrica indican que la calidad del componente está en peligro. **Análisis:** El cálculo de las métricas y la aplicación de herramientas matemáticas. **Interpretación:** La evaluación de las métricas para conocer mejor la calidad de la representación.

Uno de los problemas con es que no se comprende lo que significan realmente los datos. Es fácil malinterpretar los datos y hacer inferencias incorrectas.
Los procesos y productos para medir no están aislados de su entorno y los cambios en ese entorno invalidan las comparaciones de los datos. Los datos cuantitativos de las actividades humanas no siempre pueden tomarse como valores de entrada.

6. **Retroalimentación:** Recomendaciones derivadas de la interpretación de las métricas del producto transmitidas al equipo de software.
Las métricas del software sólo serán útiles si están caracterizadas de manera efectiva y si se validan para probar su valor.

27.5.2 Métricas de producto

Las métricas de producto se refieren a las características del software mismo. Desafortunadamente, las características del software que se miden fácilmente, como el tamaño y la complejidad ciclomática, no tienen una relación clara y consistente con los atributos de calidad como la compresión y la mantenibilidad. Las relaciones varían dependiendo de los procesos, la tecnología y el tipo de sistemas a desarrollar.
Las **métricas del producto se dividen en dos clases**:
1. Las métricas dinámicas, que son recogidas por las mediciones hechas en un programa en ejecución.
2. Las métricas estáticas, que son recogidas por las mediciones hechas en las representaciones del sistema como el diseño, el programa o la documentación.
Las **métricas dinámicas** ayudan a valorar la eficiencia y la fiabilidad de un programa y por lo general están relacionadas de forma cercana con los atributos de calidad del software. Las **métricas estáticas** ayudan a valorar la complejidad, la comprensión y la mantenibilidad de un sistema de software; por lo general están relacionadas de forma cercana con los atributos de calidad del software.
La siguiente figura describe varias métricas estáticas utilizadas para valorar los atributos de calidad.

Fan-in/Fan-out	Fan-in es una medida del número de funciones o métodos que llaman a otra función o método (por ejemplo, X). Fan-out es el número de funciones que son llamadas por una función X. Un valor alto de fan significa que X está fuertemente acoplada al resto del diseño y que los cambios en X tendrán muchos efectos importantes. Un valor alto de fan-out sugiere que la complejidad de X podría ser alta debido a la complejidad de la lógica de control necesaria para coordinar los componentes llamados.
Longitud del código	Ésta es una medida del tamaño del programa. Generalmente, cuanto más grande sea el tamaño del código de un componente, más complejo y susceptible de errores será el componente. La longitud del código ha mostrado ser la métrica más fiable para predecir errores en los componentes.
Complejidad ciclomática	Ésta es una medida de la complejidad del control de un programa. Esta complejidad del control está relacionada con la comprensión del programa. El cálculo de la complejidad ciclomática se trata en el Capítulo 22.
Longitud de los identificadores	Es una medida de la longitud promedio de los diferentes identificadores en un programa. Cuanto más grande sea la longitud de los identificadores, más probable será que tengan significado; por lo tanto, el programa será más comprensible.
Profundidad del anidamiento de las condicionales	Ésta es una medida de la profundidad de anidamiento de las instrucciones condicionales «if» en un programa. Muchas condiciones anidadas son difíciles de comprender y son potencialmente susceptibles de errores.
Índice de Fog	Ésta es una medida de la longitud promedio de las palabras y las frases en los documentos. Cuanto más grande sea el índice de Fog, el documento será más difícil de comprender.

Figura 27.12 Métricas estáticas de producto software.

Las métricas relevantes dependen del proyecto, las metas del equipo de gestión de calidad y el tipo de software a desarrollar.

PUNTOS CLAVE

- Las revisiones de los productos a entregar por el proceso del software incumben a un equipo de personas los cuales comprobarán que se han seguido los estándares de calidad, las revisiones son la técnica más utilizada para valorar la calidad.
- Las mediciones de software se utilizan para recoger datos cuantitativos acerca del software y sus procesos, los valores de las métricas de software recogidas se utilizan para hacer inferencias de la calidad del producto y del proceso.
- Las métricas de calidad del producto son de gran valor para resaltar los componentes anómalos que tienen problemas de calidad. Estos componentes se deberán analizar con más detalle.
- No existen métricas de software estandarizadas y aplicables universalmente. las organizaciones deben seleccionar métricas y analizar mediciones basadas en el conocimiento y circunstancias locales.

Gestión de la calidad

CALIDAD

Antes de iniciar las actividades de aseguramiento de la calidad del software es importante definir que es la Calidad del Software:

"Concordancia con los requisitos funcionales y de desempeño explícitamente establecidos, estándares de desarrollo explícitamente documentados y características implícitas que se espera de todo software desarrollado profesionalmente."

Es el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas que se esperan del software desarrollado profesionalmente.

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.

La calidad se refiere a características mensurables y cuando una entidad se examina en base a estas existen dos tipos de calidad:

- CALIDAD DE DISEÑO: requisitos, especificaciones y diseño del sistema.
- CALIDAD DE CONCORDANCIA: se refiere a si la implementación sigue el diseño y el sistema resultante satisface sus requisitos y metas de desempeño.

Factores de Calidad de McCall

Los factores que afectan a la calidad del software se concentran en tres aspectos importantes de un producto de software: sus características operativas, su capacidad para experimentar cambios y su capacidad de adaptarse a nuevos entornos.

OPERACIÓN DEL PRODUCTO	
Corrección	El grado en que el programa cumple con su especificación y satisface los objetivos que propuso el cliente.
Confiabilidad	El grado en que se esperaría que un programa desempeñe su función con la precisión requerida.
Eficiencia	La cantidad de código y de recursos de cómputo necesarios para que un programa realice su función.
Integridad	El grado de control sobre el acceso al software o los datos por parte de las personas no autorizadas.
Facilidad de uso	El esfuerzo necesario para aprender, operar y preparar los datos de entrada de un programa e interpretar una salida.
REVISIÓN DEL PRODUCTO	
Facilidad de mantenimiento	El esfuerzo necesario para localizar y corregir un error en un programa.
Flexibilidad	El esfuerzo necesario para modificar un programa en operación.
Facilidad de prueba	El esfuerzo que demanda probar un programa con el fin de asegurar que realiza su función.
TRANSICIÓN DEL PRODUCTO	
Portabilidad	El esfuerzo necesario para transferir el programa de un entorno de hardware o software a otro.
Facilidad de reutilización	El grado en que un programa (o partes de él) puede reutilizarse en otras aplicaciones.
Interoperabilidad	El esfuerzo necesario para acoplar un sistema con otro.

¿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

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
- Otros interesados.

Principios

- La calidad no se inyecta ni se compra, debe estar embebida.
- Es un esfuerzo de todos
- Las personas son la clave para lograrlo (Capacitación)
- Se necesita soporte gerencial (Pero se puede empezar por uno)
- Se debe liderar con el ejemplo
- No se puede controlar lo que no se mide
- Simplicidad, empezar por lo básico
- El aseguramiento de la calidad debe planificarse
- El aumento de las pruebas no aumenta la calidad
- Debe ser razonable para mi negocio

Visiones de la Calidad

- Visión del Usuario
- Visión de Manufactura
- Visión del Producto
- Visión Basada en el Valor
- Visión Trascendental

Contenidos del capítulo

27.1 Calidad de proceso y producto
27.2 Garantía de la calidad y estándares
27.3 Planificación de la calidad
27.4 Control de la calidad
27.5 Medición y métricas del software

La **calidad del software** es un concepto complejo que no es directamente comparable con la calidad de la manufactura de productos. En la manufacturación, la noción de calidad viene dada por la similitud entre el producto desarrollado y su especificación. En un mundo ideal, esta definición debería aplicarse a todos los productos, pero, para sistemas de software, existen estos **problemas**:

1. La especificación se orienta 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.
2. No se sabe cómo especificar ciertas características de calidad (por ejemplo, mantenimiento) de una forma no ambigua.
3. 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. En concreto, atributos del software como mantenibilidad, seguridad o eficiencia no pueden ser especificados explícitamente. Sin embargo, tienen un efecto importante en cómo es percibida la calidad del sistema.

Algunas personas piensan que la calidad puede lograrse definiendo **estándares y procedimientos organizacionales de calidad** que comprueban si estos estándares son seguidos por el equipo de desarrollo. Su argumento es que los

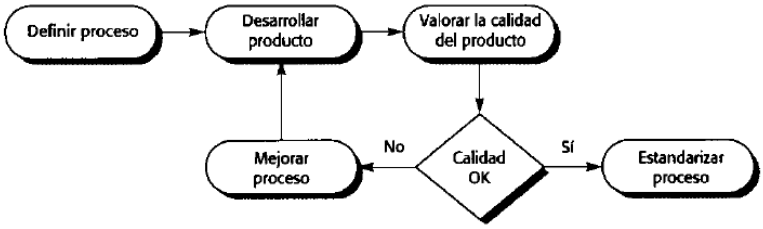
estándares deben encapsular las buenas prácticas, las cuales nos llevan inevitablemente a productos de alta calidad. En la práctica, sin embargo, es más importante la **gestión de la calidad** que los estándares y la burocracia asociada para asegurar el seguimiento de estos estándares.
Los buenos gestores aspiran a desarrollar una **«cultura de la calidad»** donde todos seamos responsables de que el desarrollo del producto sea llevado a cabo obteniendo un alto nivel de calidad en éste. Mientras estándares y procedimientos son las bases de la gestión de la calidad, los gestores de calidad experimentados reconocen que hay aspectos intangibles en la calidad del software (elegancia, legibilidad, etc.) que no puede ser incorporada en los estándares. Ellos ayudan a la gente interesada en estos aspectos intangibles de calidad y fomentan comportamientos profesionales en todos los miembros del equipo.
La gestión formal de la calidad es particularmente importante para equipos que desarrollan **sistemas grandes y complejos**. La documentación de la calidad es un registro de que es hecho por cada subgrupo en el proyecto. Esto ayuda a la gente a ver qué tareas importantes no deben ser olvidadas o que una parte del equipo no haga suposiciones incorrectas acerca de lo que otros miembros han hecho. La documentación de calidad es también un medio de comunicación sobre el ciclo de vida de un sistema. Ésta permite al grupo responsabilizarse de la evolución del sistema para saber qué ha hecho el equipo de desarrollo.
Para **sistemas pequeños**, la gestión de calidad es importante todavía, pero se debe adoptar una aproximación más informal. No son tan necesarios los documentos porque el grupo puede comunicarse informalmente. La clave de la calidad en el desarrollo de sistemas pequeños es el establecimiento de cultura de calidad y asegurarse de que todos los miembros del equipo hacen una aproximación positiva a la calidad del software.
La **gestión de calidad** del software se estructura en tres **actividades principales**:
1. Garantía de la calidad. El establecimiento de un marco de trabajo de procedimientos y estándares organizacionales que conduce a software de alta calidad.
2. Planificación de la calidad. La selección de procedimientos y estándares adecuados a partir de este marco de trabajo y la adaptación de éstos para un proyecto software específico.
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.

La **gestión de la calidad** provee una comprobación independiente de los procesos de desarrollo software. Los procesos de gestión de la calidad comprueban las entregas del proyecto para asegurarse que concuerdan con los estándares y metas organizacionales. **El equipo de garantía de calidad** debe ser independiente del equipo de desarrollo para que puedan tener una visión objetiva del software. Ellos transmitirán los problemas y las dificultades al gestor principal de la organización.

Un equipo independiente de calidad garantiza que los objetivos organizacionales y la calidad no sean comprometidos por consideraciones de presupuesto o agenda.

27.1 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 derivados.
La siguiente figura muestra una aproximación basada en proceso para conseguir la calidad del producto.



Calidad basada en procesos

Hay un vínculo claro entre la calidad del proceso y del producto en producción debido a que el proceso es relativamente fácil de estandarizar y monitorizar.
El software no se manufactura, sino que se diseña. El desarrollo de software es un proceso más creativo que mecánico. La calidad del producto, también se ve afectada por factores externos, como la novedad de una aplicación o la presión comercial para sacar un producto rápidamente.
En el desarrollo software, por lo tanto, la relación entre la calidad del proceso y la calidad del producto es muy compleja. Es difícil de medir los atributos de la calidad del software, en consecuencia, es difícil explicar cómo influyen las características del proceso en estos atributos. Además debido al papel del diseño y la creatividad en el proceso software, no podremos predecir la influencia de los cambios en el proceso en la calidad del producto.

La calidad del proceso tiene una influencia significativa en la calidad del software. La gestión y mejora de la calidad del proceso debe minimizar los defectos en el software entregado.
La gestión de la calidad del proceso implica:
1. Definir estándares de proceso.
2. Supervisar el proceso de desarrollo para asegurar que se sigan los estándares.
3. Hacer informes del proceso para el gestor del proyecto y para el comprador del software.

Un problema de la garantía de la calidad basada en el proceso es que el equipo de garantía de la calidad (QA) insista en unos estándares de proceso independientemente del tipo de software a desarrollar. El gestor principal debe intervenir para asegurar que el proceso de calidad ayude al desarrollo del producto en lugar de impedirlo.

27.2 Garantía de la calidad y estándares

La garantía de la calidad es el proceso que define cómo lograr la calidad del software y cómo la organización de desarrollo conoce el nivel de calidad requerido en el software. Como se indicó anteriormente, el proceso QA se ocupa ante todo de definir o seleccionar los estándares que deben de 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:
1. Estándares de producto. Se aplican sobre el producto software que se comienza a desarrollar. Incluyen estándares de documentación, como cabecera de comentarios estándar para definición de clases, y estándares de codificación.
2. Estándares de proceso. Definen los 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.
Los **estándares de software** son importantes por varias razones:
1. Están basadas en el conocimiento de la mejor o más apropiada práctica de la empresa, evita la repetición de errores anteriores.
2. Proveen un marco de trabajo alrededor del cual se implementa el proceso de garantía de la calidad. El control de la calidad sencillamente asegura que los estándares se siguen adecuadamente.
3. Ayudan a la continuidad cuando una persona continúa el trabajo que llevaba a cabo otra. Se reduce el esfuerzo de aprendizaje cuando se comienza un nuevo trabajo.

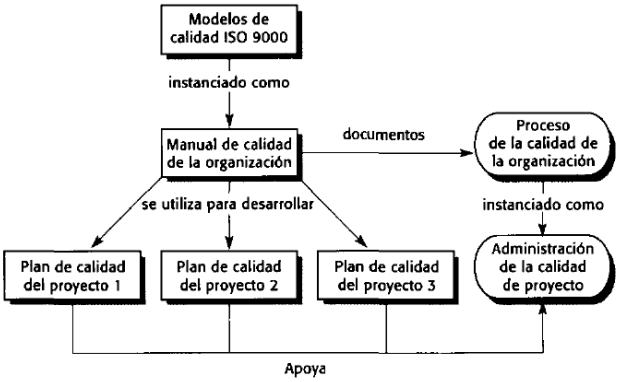
Utilizando estándares como punto de partida, el equipo de garantía de la calidad debe crear un «manual» de estándares. Éste define los estándares que son apropiados para la organización.
Algunas veces, los ingenieros de software consideran a los estándares como burocráticos e irrelevantes para las actividades técnicas de desarrollo de software.
Para evitar estos problemas, los gestores de la calidad que fijan los estándares necesitan estar informados y tomar en consideración los siguientes pasos:
1. Involucrar a los ingenieros de software en el desarrollo de los estándares del proyecto.
2. Revisar y modificar los estándares de forma regular con el fin de reflejar los cambios en la tecnología.
3. Proveer herramientas de software para apoyar los estándares donde sea necesario.

El gestor del proyecto y el gestor de calidad pueden evitarse los problemas de estándares inapropiados si planean cuidadosamente la calidad. Deben decidir cuáles son los estándares del manual que utilizarán sin cambio alguno, cuáles se modificarán y cuáles se dejarán de lado.

27.2.1 ISO 9000 Estándares de Calidad

Un sistema de Garantía de Calidad se puede definir como la estructura organizacional, responsabilidades, procedimientos, procesos y recursos para implementar la gestión de la calidad.
Para obtener el registro ISO 9001:2000, una organización de software debe establecer políticas y procedimientos para satisfacer los 20 requisitos que define el estándar para un sistema eficiente de garantía de la calidad, y además demostrar que se siguen dichas políticas y procedimientos, para lo cual la organización es sometida a auditorias periódicas.
Como el estándar es aplicable a todas las disciplinas de ingeniería, se ha desarrollados un conjunto de directrices ISO 9000-3 que ayudan interpretar el estándar para emplearlo en el proceso de SW.

Un conjunto de estándares internacionales que se puede utilizar en el desarrollo de un sistema de gestión de calidad en todas las industrias es 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 más general de estos estándares y se aplica en organizaciones interesadas en el proceso de calidad de diseño, desarrollo y mantenimiento de productos.
ISO 9001 no es un estándar específico para desarrollo de software, pero define principios generales que pueden aplicarse al software.
Aquí podemos observar las relaciones entre ISO 9000, el manual de calidad y los planes de calidad de proyectos particulares.



ISO 9000 y la gestión de la calidad

Los procesos de garantía de calidad en una organización se documentan en un manual de calidad que define el proceso de calidad.
El estándar ISO 9000 se refiere simplemente a la definición de los procedimientos que son utilizados en la compañía y la documentación asociada que muestre que los procesos han sido seguidos. Éste no se ocupa de asegurar que estos procesos sean la mejor práctica, ni asegura la calidad del producto.

27.2.2 Estándares de documentación

Los estándares de documentación en un proyecto de software son documentos muy importantes ya que son la única forma tangible de representar al software y su proceso. Los documentos estandarizados tienen una apariencia, estructura y calidad consistentes y. por lo tanto, son más fáciles de leer y de comprender.
Existen tres tipos de estándares de documentación:
1. Estándares del proceso de documentación. Definen el proceso a seguir para la producción del documento, esto implica definir los procedimientos involucrados en el desarrollo del documento y las herramientas de software utilizadas. También definen procedimientos de comprobación y refinamiento que aseguren que se produzcan documentos de alta calidad.
2. Estándares del documento. Gobiernan la estructura y presentación de los documentos.
3. Estándares para el intercambio de documentos. Aseguran que todas las copias electrónicas de los documentos sean compatibles.

Los estándares de calidad del **proceso de documentos** deben ser flexibles y les debe ser posible ajustarse a todos los tipos de documentos. Uno de los modelos posibles de proceso de documentación incluye: Crear un borrador, comprobarlo, revisarlo y rehacerlo es un proceso iterativo. Éste continúa hasta que se produce un documento de calidad aceptable. El nivel de calidad aceptable depende del tipo de documento y de los lectores potenciales de éste.
Aunque los estándares del documento se adapten a las necesidades de un proyecto específico, una buena práctica es que se utilice el mismo estilo en todos documentos producidos por una organización.
Algunos **ejemplos de estándares de documentos** a desarrollar son:
1. Estándares de identificación de documentos. Puesto que los proyectos de sistemas grandes producen cientos de documentos, cada documento debe identificarse de forma única. Para los documentos formales, este identificador es el identificador formal definido por el gestor de configuraciones. Para documentos informales, el estilo del identificador del documento es definido por el gestor del proyecto.
2. Estándares de la estructura del documento. Cada clase de documentos producida durante un proyecto de software debe seguir alguna estructura estándar.
3. Estándares de presentación de documentos. Estos estándares definen un «estilo propio » para los documentos y contribuyen notablemente a la consistencia de éstos.

4. Estándares para actualizar los documentos. Conforme el documento evoluciona y refleja los cambios en el sistema, se debe utilizar una forma consistente para indicar los cambios en el documento.

Los **estándares de intercambio** de documentos son importantes debido a que se pueden intercambiar copias electrónicas de los documentos. La utilización de estándares de intercambio permite que los documentos se transfieran electrónicamente y puedan reconstruirse en su forma original.

27.3 Planificación de la calidad

La planificación de la calidad es el proceso en el cual se desarrolla un plan de calidad para un proyecto. El plan de calidad define la calidad del software deseado y describe cómo valorar ésta. Por lo tanto, define lo que es software de «alta calidad». Sin esta definición, los diferentes ingenieros pueden trabajar en direcciones opuestas para optimizar los atributos de proyecto.
El plan de calidad selecciona los estándares organizacionales apropiados para un producto y un proceso de desarrollo particulares. Esta estructura comprende:
1. Introducción del producto. Descripción del producto, el mercado al que se dirige y las expectativas de calidad.
2. Planes de producto. Contiene las fechas de terminación del producto y las responsabilidades importantes junto con los planes para la distribución y el servicio.
3. Descripciones del proceso. Contiene los procesos de desarrollo y de servicio a utilizar para el desarrollo y administración del producto.
4. Metas de calidad. Contiene las metas y planes de calidad para el producto. Incluyendo la identificación y justificación de los atributos de calidad importantes del producto.
5. Riesgos y gestión de riesgos. Contiene los riesgos clave que podrían afectar a la calidad del producto y las acciones para abordar estos riesgos.

Plan SQA

El plan de SQA es desarrollado por un grupo de SQA (o por el equipo de desarrollo sino existe grupo SQA) y sirve como plantilla para las actividades de SQA instituidas para cada proyecto de software.
El IEEE ha recomendado un estándar para los planes de SQA cuya estructura identifica:

1. El propósito y el alcance del documento
2. Una descripción de todos los productos de trabajo de ingeniería del software.
3. Todos los estándares y prácticas aplicables que se aprovechan durante el proceso de software.
4. Acciones y tareas de SQA y su ubicación a través del proceso de software.
5. Herramientas y métodos que soportan las acciones y tareas de SQA.
6. Procedimientos de gestión de configuración de software para gestionar el cambio.
7. Métodos para ensamblar, salvaguardar y mantener todos los registros relacionados con el SQA.
8. Papeles y responsabilidades en la organización relativas a la calidad de producto.

Los planes de calidad obviamente difieren dependiendo del tamaño y del tipo de sistema que se desarrolle.
Existe una amplia variedad de **atributos de calidad del software** potenciales a considerar en el proceso de planificación de la calidad. Ver atributos de calidad en la figura siguiente.

Seguridad	**Comprensión**	**Portabilidad**
Protección	**Experimentación**	**Usabilidad**
Fiabilidad	**Adaptabilidad**	**Reutilización**
Flexibilidad	**Modularidad**	**Eficacia**
Robustez	**Complejidad**	**Aprendizaje**

Puede ser que la eficacia sea primordial, por lo que será necesario sacrificar otros factores para alcanzarla. Esto se establece en el plan, y los ingenieros que trabajan en el desarrollo deben cooperar para lograrlo.
El plan también define el proceso de evaluación de la calidad.

27.4 Control de la calidad

El control de la variación es la clave para un producto de calidad. En el contexto de software se lucha por controlar la variación en el proceso genérico que se aplica, por ejemplo: de un proyecto a otro se quiere

minimizar la diferencia entre los recursos predichos y los realmente utilizados, o se busca minimizar el número de defectos de una liberación a otra.
El control de calidad es una serie de inspecciones, revisiones y pruebas empleadas a lo largo del proceso de software para garantizar que cada producto de trabajo satisfaga los requisitos que le se le han asignado. Incluye un bucle de retroalimentación que permite afinar el proceso cuando los productos de trabajo no satisfacen sus especificaciones.

El control de la calidad implica vigilar el proceso de desarrollo de software para asegurar que se siguen los procedimientos y los estándares de garantía de calidad. En el proceso de control de calidad del proyecto se comprueba que las entregas cumplan los estándares definidos.
Existen dos enfoques complementarios que se utilizan para **comprobar la calidad de las entregas** de un proyecto:
1. Revisiones de la calidad donde el software, su documentación y los procesos utilizados en su desarrollo son revisados por un **grupo de personas**. Se encargan de comprobar que se han seguido los estándares del proyecto y el software y los documentos concuerdan con estos estándares. Se toma nota de las desviaciones de los estándares y se comunican al gestor del proyecto.
2. Valoración automática del software en la que el software y los documentos producidos se procesan por algún **programa** y se comparan con los estándares que se aplican a ese proyecto de desarrollo en particular. Esta valoración automática comprende una medida cuantitativa de algunos atributos del software.

27.4.1 Revisiones de la calidad

Las revisiones son el método más utilizado para validar la calidad del un proceso o de un producto. Involucran a un grupo de personas que examinan todo o parte del proceso software, los sistemas o su documentación asociada para descubrir problemas potenciales. Las conclusiones de la revisión se registran formalmente y se pasan al autor o a quien sea responsable de corregir los problemas descubiertos.
En la siguiente figura se listan los **tipos de revisiones**.

Inspecciones de diseño o programas	**Detectar errores finos en los requerimientos, el diseño o el código. La revisión es conducida por una lista de verificación de los posibles errores.**
Revisiones del progreso	**Proveer información del progreso del proyecto útil para su gestión. Ésta es una revisión tanto del proceso como del producto y se refiere a costes, duración y planificación.**
Revisiones de la calidad	**Llevar a cabo un análisis técnico de los componentes del producto o documentación para encontrar diferencias entre la especificación y el diseño del componente, código y documentación, y para asegurar que se siguen los estándares de calidad definidos.**

Tipos de revisiones

El equipo de revisiones debe tener un núcleo de tres o cuatro personas como revisores principales.
Uno debe ser el diseñador principal, el cual tendrá la responsabilidad de tomar las decisiones técnicas. Los revisores principales pueden invitar a otros miembros del proyecto, para que colaboren en la revisión.
Los documentos a revisar deben distribuirse con anterioridad a la revisión para dar tiempo a los revisores a que los lean y los comprendan.
La revisión misma es relativamente corta (dos horas a lo más). Un miembro del equipo preside la revisión y otro registra formalmente todas las decisiones de la revisión. Este registro pasará a formar parte de la documentación formal del proyecto. El presidente es responsable de asegurar que se hagan todos los cambios requeridos. Si se requieren cambios importantes, habrá que hacer un seguimiento posterior de la revisión.

COSTO DE LA CALIDAD

Incluye todos los costos que genera la búsqueda de calidad o que demanda el desarrollo de las actividades relacionadas con la calidad. Estos de dividen en:

- Costos de Prevención: planificación, revisiones técnicas formales, equipo de prueba y entrenamiento.
- Costos de Evaluación: inspección en el proceso y entre procesos, calibración y mantenimiento de equipos y prueba.
- Costos de fallas: son aquellos que desaparecerían sino aparecieran defectos antes de enviar un producto a los clientes. Se subdividen en:
  - Costo de Fallas Internas: aparecen cuando se detectan defectos en el producto antes del envío. Incluyen reelaboración, reparación y análisis en modo de fallas.

- Costo de Fallas Externas: se asocian a defectos detectados después del envío del producto al cliente. Ejemplo: resolución de quejas, devolución y reemplazo del producto, soporte de ayuda en línea y trabajo de garantía.

CALIDAD Y PROCESO DE DESARROLLO

El proceso es el único factor controlable al mejorar la calidad del software y su rendimiento como organización.

Proceso: es la secuencia de pasos ejecutados para un propósito dado (IEEE).
Proceso de Software: Un conjunto de actividades, métodos, prácticas, y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados. Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse. Debe ser explícitamente modelado si va a ser administrado.

¿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

¿Qué ocurre cuando el proceso no está definido?

Las actividades del proceso no están definidas:

- no se pueden planificar, controlar ni asegurar los resultados

¿Qué cosas ocurren frecuentemente en los proyectos de desarrollo de software?

- Atrasos en las entregas
- Costos Excedidos
- Falta cumplimiento de los compromisos
- No están claros los requerimientos
- El software no hace lo que tiene que hacer
- Trabajo fuera de hora
- Fenómeno del 90-90
- ¿Adónde está ese componente?

Criterio para la Definición de un Proceso

Este elemento del proceso...	Responde la pregunta básica:
Propósito	¿Por qué se realizó un proceso?
Entrada	¿Qué productos de trabajo se utilizan?
Salida	¿Qué productos de trabajo se generan?
Rol	¿Quién (o que) realiza las actividades?
Actividad	¿Qué se hace?
Criterio de Entrada	¿Cuándo (bajo que circunstancias) pueden iniciarse los procesos?
Criterio de Salida	¿Cuándo (bajo que circunstancias) pueden los procesos considerarse completos?
Procedimiento	¿Cómo son implementadas las actividades?

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 o herramientas.

GESTIÓN DE CALIDAD

La Gestión de Calidad es una actividad que se aplica a lo largo de todo el proceso de software, y abarca:

1. Un proceso de Garantía de la calidad del software (SQA)
2. Tareas específicas de aseguramiento y control de calidad (que incluyen revisiones técnicas formales y una estrategia de pruebas de varios niveles)
3. Prácticas efectivas de ingeniería de software (métodos y herramientas)
4. Control de todos los productos de trabajo del software y los cambios que generan
5. Un procedimiento para garantizar la concordancia con los estándares de desarrollo del software (cuando sea aplicable)
6. Mecanismos de medición e informe

Garantía o Aseguramiento de la Calidad del Software (SQA)

Es un patrón sistemático y planificado de acciones que se requiere para garantizar alta calidad en el software. Es un conjunto de funciones de auditoría e información que evalúan la efectividad y que tan completas son las actividades de control de calidad.

La garantía de calidad involucra a:

- Los ingenieros de software que realizan el trabajo técnico (revisiones y pruebas)
- Un grupo SQA que tiene la responsabilidad de planificar, supervisar, guardar registros, analizar y reportar la garantía de calidad.

El SEI (Software Engineering Institute) recomienda un conjunto de actividades a ser realizadas por un grupo SQA independiente:

a) Preparar un Plan SQA para un proyecto→ identifica las evaluaciones, auditorías y revisiones a realizar, los estándares aplicables al proyecto, los procedimientos para el informe y seguimiento de errores, documentos a producir y la retroalimentación proporcionada al equipo de proyecto .
b) Participar en el desarrollo de la descripción del proceso de software del proyecto. → revisa que el proceso concuerde con las políticas organizacionales, estándares internos y externos (ej.: ISO 9001) y otras partes del Plan de Proyecto del Software.
c) Revisar las actividades de ingeniería del software para verificar que se ajusten al proceso software definido → identifica, documenta y sigue las desviaciones del proceso y verifica que se hayan hecho las correcciones.
d) Audita productos de trabajo de software seleccionados para verificar que se ajusten con los definidos como parte del proceso de software.
e) Garantiza que las desviaciones en el trabajo del software y en los productos de trabajo estén documentadas y se manejen de acuerdo con el procedimiento establecido.

f) Registrar cualquier falta de ajuste y lo informa al gestor ejecutivo

Además de estas actividades, el grupo SQA coordina el control y la gestión del cambio y ayuda a recopilar y analizar métricas de software.

Revisiones

Las revisiones se aplican en varios puntos durante la ingeniería del software y sirven para descubrir errores y defectos que luego pueden eliminarse. Existen muchos tipos, de los cuales se consideran el medio más efectivo para descubrir errores y mejorar la calidad del SW la llamada Revisión Técnica Formal o Inspección de código.

Revisiones Técnicas Formales (RTF)

Es una actividad de control de calidad del software lleva a cabo por una Junta de Revisión que incluye recorridos, inspecciones, revisiones cíclicas y otro pequeño grupo de evaluaciones técnicas de software. Una RTF sólo tendrá éxito si se planifica, controla y atiende apropiadamente.

Su objetivo principal es descubrir los errores durante el proceso, de modo que no se conviertan en defectos después de liberar el software. Otros objetivos son:

a) Descubrir errores en la función, lógica o implementación en cualquier representación del software.
b) Verificar que el software en revisión satisface sus requisitos.
c) Garantizar que el software se ha representado de acuerdo con los estándares predefinidos.
d) Lograr software desarrollado en una manera uniforme
e) Hacer proyectos más manejables.

El beneficio de las RTF es el descubrimiento temprano de errores, evitando que los mismos se propaguen al paso siguiente en el proceso de software y reduciendo sustancialmente los costos de las actividades subsecuentes.

En el Modelo de Amplificación de Defectos, durante un paso, **nuevos errores** se pueden generar de manera inadvertida. La revisión puede fallar en descubrir errores generados de manera reciente y errores de pasos anteriores, lo que resulta en un cierto número de **errores que se pasan por alto**. Algunos de los **errores pasados por alto desde pasos anteriores se amplifican** con el trabajo actual (factor de amplificación x). Las subdivisiones de los recuadros representan estas características y el porcentaje de eficiencia de la detección de errores, una minuciosidad de la revisión.

Junta de revisión

Cualquier reunión de revisión debe acogerse a las siguientes restricciones:

- Deben involucrarse para la revisión (normalmente) entre tres y cinco personas;
- Se debe preparar con anticipación, pero sin que requiera más de dos horas de trabajo a cada persona;
- La duración de la reunión de revisión debe ser menor de dos horas.

Es por esto que cada RTF centra su atención en un producto de trabajo, es decir, en una parte específica y pequeña del software total, con lo que la probabilidad de descubrir errores es mayor. Ejemplo: una porción de una especificación de requisitos, un diseño detallado del módulo, un listado del código fuente de un módulo).

El proceso es el siguiente:

a) El individuo que ha desarrollado el producto -el productor- informa al jefe del proyecto de que el producto está terminado y que se requiere una revisión.
b) El jefe del proyecto contacta con un jefe de revisión, que evalúa la disponibilidad del producto, genera copias del material del producto y las distribuye a dos o tres revisores para que se preparen por adelantado.
c) Cada revisor estará entre una y dos horas revisando el producto, tomando notas y también familiarizándose con el trabajo.
d) De forma concurrente, también el jefe de revisión revisa el producto y establece una agenda para la reunión de revisión que, normalmente, queda convocada para el día siguiente.

e) La reunión de revisión es llevada a cabo por el jefe de revisión, los revisores y el productor. Uno de los revisores toma el papel de registrador de forma escrita) de todos los sucesos importantes que se produzcan durante la revisión.
f) La RTF comienza con una explicación de la agenda y una breve introducción a cargo del productor. Entonces el productor procede con el «recorrido de inspección>>del producto, explicando el material, mientras que los revisores exponen los problemas que descubrieron antes de la junta. Cuando se descubren problemas o errores válidos, el registrador los va anotando.
g) Al final de la revisión, todos los participantes en la RTF deben decidir si:
    - Aceptan el producto sin posteriores modificaciones;
    - Rechazan el producto debido a los serios errores encontrados (una vez corregidos, ha de hacerse otra revisión).
    - Aceptan el producto provisionalmente (se han encontrado errores menores que deben ser corregidos, pero sin necesidad de otra posterior revisión).
h) Una vez tomada la decisión, todos los asistentes llenan un documento de finalización en el que indican su participación en la RTF y su conformidad con los hallazgos y conclusiones del equipo de revisión.

Informe de la revisión y conservación de registros

En base a las anotaciones del Registrador se genera, al finalizar la junta, una Lista de problemas de revisión y se llena un Informe Resumen de la RTF.

El Informe Resumen de la RTF consta de una sola página (con posibles anexos) que responde a tres preguntas:

1. ¿Qué fue revisado?
2. ¿Quién lo revisó?
3. ¿Cuáles fueron los hallazgos y conclusiones?

Este se adjunta al registro histórico del proyecto y puede ser enviada al jefe del proyecto y a otras partes interesadas.

La Lista de problemas de revisión sirve para dos propósitos:(1) identificar áreas problemáticas dentro del producto y (2) servir como lista de comprobación de puntos de acción que guíe al productor para hacer las correcciones. Normalmente se adjunta al informe resumen.

Es importante establecer un procedimiento de seguimiento que asegure los problemas son corregidos adecuadamente. Un enfoque consiste en asignar la responsabilidad del seguimiento al jefe de revisión.

Directrices de la revisión

Se deben establecer de antemano directrices para conducir las revisiones técnicas formales, distribuyéndolas después entre los revisores para ser consensuadas y,finalmente, seguidas.

Conjunto mínimo de directrices para las revisiones técnicas formales:

- *Revisar el producto, no al productor:* Una RTF involucra gente y egos. Conducida adecuadamente, todos los participantes deben tener un sentimiento agradable de estar cumpliendo su deber. Se deben señalar los errores educadamente; el tono de la reunión debe ser distendido y constructivo.
- *Establecer una agenda y respetarla*: El jefe de revisión tiene la responsabilidad de mantener el plan de la reunión y no vacilar en llamar la atención de la gente cuando se empiece a divagar.
- *Limitar el debate y la impugnación:* cuando no existe consenso sobre el impacto de un problema planteado por un revisor, este debe ser registrado y dejado su tratamiento para después.
- *Enunciar áreas de problemas, sin intentar resolver todos los que se hayan señalado:* una revisión no es una sesión para resolver problemas, esto debe posponerse para después de la junta de revisión.
- *Tomar notas:* es bueno que el registrador tome notas en un pizarrón, de modo que las palabras y prioridades puedan ser evaluadas por los otros revisores mientras se registra la información.
- *Limitar el número de participantes e insistir en la preparación anticipada:* el número de miembros del equipo de revisión debe ser el mínimo necesario y estos deben prepararse por anticipado. El jefe de revisión debe solicitar comentarios escritos.

- *Desarrollar una lista de verificación para cada producto que tenga la probabilidad de ser revisado:* esta ayuda al jefe de revisión a estructurar la junta y a los revisores a enfocarse en los problemas importantes.
- *Asignar recursos y programar las RTF: las revisiones son efectivas si se programan como parte del proceso de desarrollo, incluyendo el tiempo necesario para realizar las modificaciones que resultan de la RTF.*
- *Realizar un entrenamiento significativo de los revisores: el entrenamiento debe ser formal y enfocarse tanto a los problemas del proceso, como al lado psicológico y humano de as revisiones.*
- *Analizar las revisiones previas:* para descubrir problemas en el proceso de revisión mismo

Ya que son muchas las variables involucradas que inciden en una revisión provechosa, cada organización debe experimentar para determinar que enfoque funciona mejor en un contexto local.

Revisiones basadas en muestras

Las revisiones implican recursos y tiempo que son limitados en el proyecto, de manera tal que si el proyecto se encuentra retrasado suele disminuirse el tiempo dedicado a la revisión, lo que conduce a una falta de calidad. Una solución es utilizar *revisiones basadas en muestras.*

Este proceso de revisión se basa en tomar muestras de todos los productos de trabajo con el objeto de cuantificar aquellos que sean más proclives al error y concentrar los recursos disponibles en revisar dichos productos.

Pasos:

1. Inspeccionar una fracción *a1* de cada producto de trabajo de software *i*. Registrar el nº de fallas *fi* encontradas dentro de *ai*.
2. Desarrollar una estimación bruta del nº de fallas del producto calculando: *fi*1/ai*.
3. Ordenar los productos de trabajo en forma descendente de acuerdo con la estimación bruta del número de fallas en cada uno.
4. Enfocar los recursos de revisión disponibles en aquellos productos de trabajo con el mayor nº estimado de fallas.

La fracción con la que se hizo el muestreo debe ser representativa del producto de trabajo como un todo y ser lo suficientemente grande de tal manera que sea significativa para los revisores que realicen el muestreo.

Garantía de la Calidad Estadística del Software

La garantía de la calidad estadística refleja una tendencia creciente en la industria de adoptar un enfoque más cuantitativo acerca de la calidad. Implica los siguientes pasos:

1. Se agrupa y se clasifica la información sobre los defectos del software.
2. Se intenta encontrar la causa subyacente de cada defecto. Por ejemplo:
   - Especificación incompleta o errónea (EIE).
   - Mala interpretación de la comunicación del cliente (MCC).
   - Desviación deliberada de la especificación (DDE).
   - Incumplimiento de los estándares de programación (IEP).
   - Error en la representación de los datos (ERD).
   - Interfaz de componente inconsistente (IMI).
   - Error en la lógica de diseño (ELD).
   - Prueba incompleta o errónea (PIE).
   - Documentación imprecisa o incompleta (DII).
   - Error en la traducción del diseño al lenguaje de programación. (TLP)
   - Interfaz hombre-computadora ambigua o inconsistente (IHC)

3. Mediante el principio de Pareto (el 80 % de los defectos se pueden encontrar en el 20% de todas las posibles causas), se aísla el 20 % (los vitales). Ejemplo: en la siguiente tabla observamos que EIE, MCC y ERD son las causas vitales que explican el 53 % de todos los defectos para un proyecto xxx.

4. Una vez que se han identificado los defectos vitales, se actúa para corregir los problemas que han producido los defectos. Conforme sus causas se corrigen, nuevas causas ocuparán la parte superior de la clasificación y deberán ser corregidas. Para el ejemplo anterior, para corregir MCC, el desarrollador de software puede implementar técnicas que faciliten la recopilación de requisitos para mejorar la calidad de la comunicación y las especificaciones del cliente.

Fiabilidad del Software

La fiabilidad del software, a diferencia de otros factores de calidad, se puede medir, dirigir y estimar mediante datos históricos o de desarrollo.
La fiabilidad del software se define en términos estadísticos como la probabilidad de operación libre de fallas de un programa de computadora en un entorno determinado y durante un tiempo específico.
Una falla es cualquier falta de concordancia con los requisitos del software.

Medidas de fiabilidad y disponibilidad

Todavía se debate sobre la relación entre los conceptos de la fiabilidad del hardware y su aplicabilidad al software, pues la mayoría de los modelos de fiabilidad relativos al hardware están más orientados a fallas debido al uso que a fallas debidas a defectos de diseño, mientras que todas las fallas de software se originan por problemas de diseño. Sin embargo existen algunos conceptos que se aplican a ambos sistemas:

Considerando un sistema basado en computadora, una sencilla medida de la fiabilidad es el tiempo medio entre fallas (TMEF), donde;
TMEF = TMDF + TMDR
Las siglas TMDF y TMDR corresponden a tiempo medio de fallo y tiempo medio de reparación, respectivamente.

Otra medida es la de Disponibilidad, que es la probabilidad de que un programa opere de acuerdo con los requisitos en un momento dado, y se define como:
Disponibilidad = [TMDF/ (TMDF + TMDR)] x 100 %

La medida de fiabilidad TMEF es igualmente sensible al TMDF que al TMDR, mientras que la medida de disponibilidad es algo más sensible al TMDR, una medida indirecta de la facilidad de mantenimiento del SW.

Seguridad del software

Este enfoque es similar al análisis de riesgos, siendo la principal diferencia su énfasis en los conflictos tecnológicos más que en los tópicos relacionados con el proyecto.

La seguridad del software es una actividad de garantía de calidad del software que se centra en la identificación y evaluación de los riesgos potenciales que pueden producir un impacto negativo en el software y hacer que falle el sistema completo. Si se pueden identificar pronto los riesgos en el proceso de ingeniería del software podrán especificarse las características del diseño del software que permitan eliminar o controlar los riesgos potenciales.
Inicialmente, se identifican los riesgos y se clasifican por su importancia y su grado de riesgo.
Cuando se han identificado estos riesgos del sistema, se utilizan técnicas de análisis (como el análisis del árbol de fallos, la lógica de tiempo real o los modelos de redes de Petri) para asignar su gravedad y su probabilidad de ocurrencia. Para que sea efectivo, se tiene que analizar el software en el contexto del sistema completo.

Una vez que se han identificado y analizado los riesgos, se pueden especificar los requisitos relacionados con la seguridad para el software a través de una lista de eventos no deseables y las respuestas del sistema a estos eventos.

Aunque la fiabilidad y la seguridad del software están bastante relacionadas, es importante entender la sutil diferencia que existe entre ellas. La fiabilidad del software utiliza el análisis estadístico para determinar la probabilidad de que pueda ocurrir un fallo del software.
Sin embargo, la ocurrencia de un fallo no lleva necesariamente a un riesgo o a un accidente. La seguridad del software examina los modos según los cuales los fallos producen condiciones que pueden llevar a accidentes. Es decir, los fallos se evalúan en el contexto de un completo sistema basado en computadora.

PUNTOS CLAVE

- La gestión de la calidad del software permite señalar si éste tiene un escaso número de defectos y si alcanza los estándares requeridos de mantenibilidad, fiabilidad, portabilidad, etcétera, las actividades de la gestión de la calidad comprenden la garantía de la calidad que establece los estándares para el desarrollo de software, la planificación de la calidad y el control de la calidad que comprueba el software con respecto a los estándares definidos.
- Un manual de calidad organizacional debe documentar un conjunto de procedimientos de garantía de la calidad. Éste puede basarse en los modelos genéricos sugeridos en los estándares ISO 9000.
- Los estándares de software son importantes para garantizar la calidad puesto que representan una identificación de las «mejores prácticas». El proceso de control de calidad implica comprobar que el proceso del software y el software a desarrollar concuerdan con estos estándares.

Mejora de procesos: **INTRODUCCIÓN**

La mejora de procesos significa entender los procesos existentes y optimizarlos para mejorar la calidad del producto y. en particular, en reducir el número de defectos en el software entregado. Una vez que se logra esto, las metas principales son la reducción de costes y tiempo de desarrollo.
Los procesos del software son intrínsecamente complejos y comprenden un gran número de actividades. Sus atributos son entre otros:

Comprensión	**¿Hasta qué punto se define completamente el proceso y cómo de fácil es comprender su definición?**
Visibilidad	**¿Las actividades del proceso culminan en resultados claros de forma que el progreso del proceso es visible externamente?**
Apoyo	**¿Hasta qué punto las actividades del proceso pueden apoyarse en herramientas case?**
Aceptación	**¿El proceso definido es aceptable y utilizable por los ingenieros responsables de producir el software?**
Fiabilidad	**¿El proceso diseñado es de tal forma que los errores del proceso se evitan o identifican antes de que se conviertan en errores de producto?**
Robustez	**¿Puede continuar el proceso a pesar de los problemas inesperados?**
Mantenibilidad	**¿Puede evolucionar el proceso para reflejar los requerimientos organizacionales cambiantes o las mejoras identificadas del proceso?**
Rapidez	**¿Cómo de rápido se puede completar el proceso de construcción de un sistema a partir de una especificación dada?**

No es posible hacer mejoras de procesos que optimicen todos los atributos del proceso de forma simultánea.
La mejora de procesos no significa simplemente adoptar métodos o herramientas particulares o utilizar algún modelo de un proceso utilizado en otro lugar, siempre existen factores organizacionales particulares, procedimientos y estándares que influyen en el proceso, por lo que la mejora de procesos es una actividad específica de una organización o de parte de ella si es una organización grande.

La mejora de procesos es:

- Una actividad cíclica: involucra tres estados.
  - Medición: Proceso de medición de los atributos del proyecto actual o del producto.
  - 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.
  - Cambio: Introducción de los cambios del proceso identificados en el análisis.
- Es una actividad a largo plazo: cada etapa puede durar varios meses.
- Es una actividad continua: el entorno de negocio cambia y los procesos evolucionan para tener en cuenta estos cambios o se introducen nuevos procesos,.

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.

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, estableciendo un lenguaje común en la Organización.

- Se produce un cambio organizacional y cultural.
- Se adoptan Técnicas y Métodos.

Modelos para la Mejora de Procesos

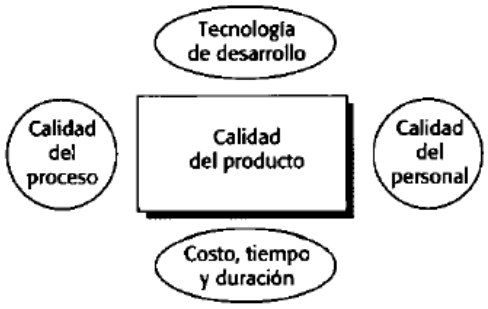
Algunos son:

- SPICE: Software Process Improvment Capability Evaluation.
- IDEAL: Initiating, Diagnosing, Establishing, Acting, Leveraging.

CALIDAD DE PRODUCTO Y DE PROCESO

La mejora de procesos está basada en la suposición de que la calidad del proceso de desarrollo es crítica para la calidad del producto. Estas nociones de mejora de procesos provienen del ingeniero estadounidense W. E. Deming, quien trabajó en la industria japonesa después de la Segunda Guerra Mundial para mejorar la calidad aplicando un control estadístico de la calidad. Éste se basa en medir el número de defectos en los productos y relacionar estos defectos con el proceso. Éste se mejora con el propósito de lograr un proceso repetible, es decir, hasta que los resultados del proceso sean predecibles y el número de defectos se reduzca. Después se estandariza e inicia un ciclo de mejoras de calidad.

Estos conceptos son aplicables al software, aunque hay diferencias importantes:
Cuando existe manufactura, la relación entre proceso y producto es obvia. Este vínculo es menos para un producto intangible y dependiente de un proceso de diseño en el que las capacidades del individuo son importantes, como lo es el software.
Para los productos software existen cuatro factores que afectan a la calidad del producto, cuya influencia depende del tamaño y del tipo de proyecto:



Para sistemas muy grandes, el determinante principal de la calidad del producto es el proceso del software. Los problemas principales son la integración, la gestión y las comunicaciones.
Si los equipos son pequeños, es importante contar con una buena tecnología de desarrollo que favorezca la productividad del equipo.
La calidad del producto también se ve afectada si un proyecto, independientemente de su tamaño, está mal presupuestado o planificado con un tiempo de entrega irreal.

A menudo, la causa real de los problemas en la calidad del software es el hecho de que las organizaciones deben competir para sobrevivir, para lo cual infravaloran el esfuerzo o prometen una entrega rápida con el fin de conseguir el contrato de desarrollo y luego sacrifican la calidad en un intento de mantener estos compromisos.

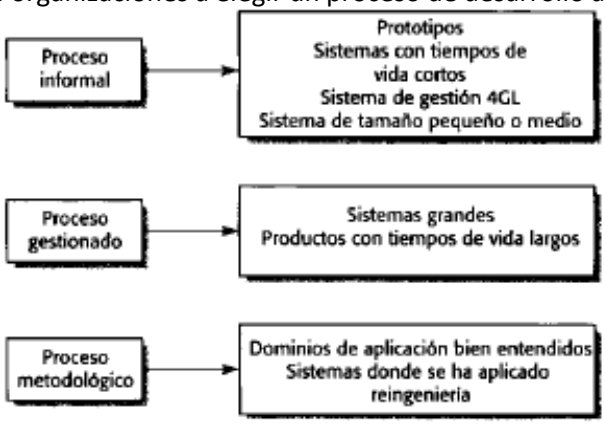
CLASIFICACIÓN DE LOS PROCESOS

Hay cuatro clases de procesos software:

1. *Procesos informales.* Son procesos en los que no existe un modelo de proceso definido de forma estricta. Los procesos informales podrían utilizar procedimientos formales, pero los procedimientos a utilizar y sus relaciones son definidos por el equipo de desarrollo.
2. *Procesos gestionados.* Se utiliza un modelo de proceso para dirigir el proceso de desarrollo. El modelo de proceso define los procedimientos, su agenda y las relaciones entre los procedimientos.

3. *Procesos metodológicos*. Se utiliza algún o algunos métodos de desarrollo definidos (como los métodos sistemáticos para diseño orientado a objetos). Estos procesos se benefician de la existencia de herramientas CASE para el diseño y el análisis.
4. *Procesos de mejora.* Son procesos que tienen inherentemente objetivos de mejora. Existe un presupuesto específico para estos procesos de mejora, y de procedimientos para introducir tales mejoras. Como parte de estas mejoras, se introducen mediciones cuantitativas del proceso.

Estas clasificaciones se solapan, por lo que un proceso puede ser de diferentes clases.
Esta clasificación es útil debido a que sirve como base para la mejora de procesos multidimensional y ayuda a las organizaciones a elegir un proceso de desarrollo apropiado para los diferentes productos:



La clasificación de procesos reconoce que el proceso afecta a la calidad del producto. Sin embargo no supone que el proceso sea siempre el factor dominante. Provee una base para mejorar los procesos, aplicando diferentes tipos de mejora de procesos a diferentes tipos de procesos. Por ejemplo, las mejoras para los procesos metodológicos se basan en mejores métodos de capacitación, mejor integración de los requerimientos y el diseño, etc.
La mayoría de los procesos software tienen ahora el apoyo de herramientas CASE por lo que son procesos con soporte. Sin embargo, los procesos pueden tener otras clases de herramientas de apoyo independientemente de que se utilice o no un método de diseño estructurado. La efectividad de las herramientas de apoyo depende del tipo de procesos utilizado. Las herramientas CASE para análisis y diseño son las más efectivas cuando se sigue un proceso metodológico. Las herramientas especializadas dan soporte a actividades individuales.

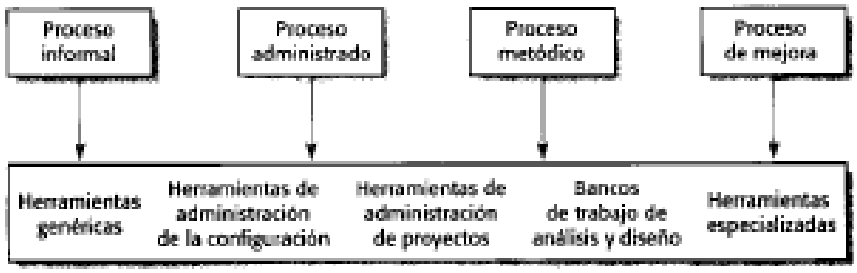


Figura 28.5
Herramientas de apoyo al proceso.

MEDICIÓN DEL PROCESO

La medición de los atributos de proceso y de producto es esencial para la mejora de procesos.
Las métricas de proceso se utilizan para evaluar si la eficiencia de un proceso ha mejorado, pero no se pueden utilizar para determinar si la calidad del producto ha mejorado. Las métricas de producto también deben hacerse y relacionarse con las actividades del proceso.

Se pueden recoger tres clases de métricas de proceso:
l. El tiempo requerido para completar un proceso en particular (calendario)

2. Los recursos requeridos para un proceso en particular (esfuerzo total en personas/día, los costes de viajes. los recursos de cómputo, etcétera.)
3. El número de ocurrencias de un evento en particular. Ejemplo: el número de defectos descubiertos durante la inspección del código, el número de peticiones de cambios en los requerimientos, etcétera.

Los dos primeros tipos de mediciones se utilizan para ayudar a descubrir si los cambios en el proceso mejoran la eficiencia de un proceso. Por ejemplo: se mide el tiempo y esfuerzo requeridos para moverse de un punto a otro del proceso, utilizando los valores medidos para indicar áreas donde se puede mejorar el proceso. Después de introducir los cambios, las mediciones de los atributos del sistema muestran si los cambios en el proceso han sido beneficiosos para reducir el tiempo o esfuerzo requerido.
Las mediciones del número de eventos que ocurren tienen una influencia directa en la calidad del software. Por ejemplo, incrementar el número de defectos descubiertos al cambiar el proceso de inspección del programa probablemente se reflejará en una mejora de la calidad del producto. Sin embargo, esto tiene que confirmarse por mediciones posteriores del producto.

La dificultad fundamental en la medición del proceso es conocer qué medir. Basili y Rombach propusieron el paradigma denominado GQM (Goal-QuestionMetric).
Este enfoque se basa en la identificación de:

1. METAS: Lo que la organización está tratando de lograr. Ejemplo: Reducir los tiempos de desarrollo del producto
2. PREGUNTAS: Son refinamientos de las metas en las que se identifican las áreas específicas de incertidumbre relacionadas con las metas. Normalmente, una meta tendrá varias preguntas asociadas que requieren respuesta. Ejemplo: ¿Cómo se puede reducir el tiempo necesario para finalizar los requerimientos del producto?
3. MÉTRICAS. Son mediciones que hay que recoger para ayudar a responder las preguntas y confirmar si las mejoras del proceso ayudaron a cumplir la meta deseada. Ejemplo: Medir el número de comunicaciones formales entre el cliente y el proveedor para cada cambio de los requerimientos.

La ventaja de este enfoque cuando se aplica a la mejora de procesos es que separa las cuestiones organizacionales (las metas) de las cuestiones específicas del proceso (las preguntas). Se centra en la recolección de datos y señala que estos datos se deben analizar de diferentes formas, dependiendo de la pregunta que se pretenda contestar.
En el método ami de mejora de procesos del software el enfoque GQM se desarrolló y combinó con el modelo de madurez de la capacidad del SEI. Los desarrolladores del método ami proponen un enfoque de etapas para la mejora de procesos en la que las mediciones se incluyen después de que la organización haya introducido algún tipo de disciplina en sus procesos.

ANÁLISIS Y MODELADO DE PROCESOS

Consiste en estudiar los procesos existentes y desarrollar un modelo abstracto de estos procesos que capte sus características principales. Estos modelos ayudarán a comprender el proceso y a comunicarlo a otros.
Los modelos formales sirven como un punto de inicio útil para el análisis de procesos. Este estándar define las actividades críticas y el ciclo de vida de los productos a entregar que se deben producir .Sin embargo, raramente incluyen suficiente detalle o reflejan las actividades reales del desarrollo de software.

Las técnicas de análisis de procesos comprenden:
1. Cuestionarios y entrevistas. A los ingenieros que trabajan en un proyecto se les pregunta sobre lo que sucede realmente. Las respuestas de un cuestionario formal se refinan durante las entrevistas personales con los involucrados en el proceso.
2. Estudios etnográficos. Se utilizan para comprender la naturaleza del desarrollo de software como una actividad
Humana, revelando sutilezas y complejidades no descubiertas por otras técnicas.

Cada uno de estos enfoques tiene ventajas y desventajas.

El análisis basado en cuestionarios se lleva a cabo rápidamente una vez que se han diseñado las preguntas, pero si estás no están bien redactadas o son inapropiadas, o los encuestados dan las respuestas que creen que el encuestador desea escuchar, el resultado será un modelo incompleto o impreciso del proceso. Las entrevistas con la gente involucrada en el proceso son más flexibles que los cuestionarios, pudiendo adaptar el guión de preguntas previamente preparado según las respuestas que se espera obtener de las diferentes personas.
El análisis etnográfico es más apropiado para descubrir los procesos que realmente se utilizan. Sin embargo, es una actividad costosa y a largo plazo que puede durar por lo menos varios meses. Se basa en la observación externa del proceso desde las primeras etapas del proyecto hasta la entrega y mantenimiento del producto, por lo que no es práctico en proyectos largos que duran varios años.

Después del análisis, los procesos se describen utilizando un modelo de procesos.
Los modelos de proceso son vistas genéricas o simplificadas de los procesos de software, donde se muestran las actividades y las salidas del proceso. Estos modelos genéricos tienen diferentes instancias, dependiendo del tipo de software a desarrollar y del entorno organizacional.
Los modelos de procesos genéricos son una base útil para analizar los procesos. Sin embargo, no incluyen suficiente información para el análisis y mejora de procesos. Es necesario un modelo de procesos detallado que indique:

Actividad (representada por un rectángulo redondeado sin sombra)	Una actividad tiene claramente definidos un objetivo, las entradas y las condiciones de salida. Ejemplos de actividades son preparar un conjunto de datos de prueba para probar un módulo, codificar una función o módulo, hacer la prueba de lectura a un documento, etcétera. Por lo general, una actividad es atómica, es decir, incumbe a una persona o a un grupo. No se descompone en subactividades.
Proceso (representado por un rectángulo redondeado con sombra)	Un proceso es un conjunto de actividades que tiene alguna coherencia , cuyo objetivo, por lo general, se acuerda dentro de la organización. Ejemplos de procesos son el análisis de requerimientos, el diseño arquitectónico, la planificación de pruebas, etcétera.
Producto a entregar (representado por un rectángulo con sombra)	Un producto a entregar es una salida tangible de una actividad prevista en un plan de proyecto.
Condición (representada por un paralelogramo)	Una condición es o una precondición que debe cumplirse antes de iniciar un proceso o actividad o una postcondición que se cumple después de terminar el proceso o la actividad.
Rol (representado por un círculo con sombra)	Un rol es un área limitada de responsabilidad. Ejemplo de roles son el gestor de configuraciones, el ingeniero de pruebas, el diseñador de software, etc. Una persona puede tener varios roles distintos, y un solo rol se puede asociar a varias personas.
Excepción (no se muestra en los ejemplos pero se representa como un cuadro de doble borde)	Una excepción es una descripción de cómo modificar el proceso si ocurre un evento anticipado o no anticipado. Las excepciones a menudo son indefinidas y se deja a la habilidad de los administradores e ingenieros del proyecto el manejo de la excepción.
Comunicación (representada por una flecha)	Un intercambio de información entre personas o entre personas y sistemas de cómputo de apoyo. Las comunicaciones pueden ser informales o formales. Las comunicaciones formales podrían ser que un administrador del proyecto apruebe un producto a entregar; las comunicaciones informales podrían ser el intercambio de un correo electrónico para resolver las ambigüedades en un documento.

Figura 28.6 Elementos de un modelo de proceso.

Los modelos de proceso detallados son sumamente complejos, de manera que en lugar de tener toda la información en un solo modelo, se necesita hacer varios modelos a diferentes niveles de abstracción. Ejemplo: un modelo sobre las actividades y otro con la información de control que dirige la ejecución del proceso.

Excepciones del proceso

Aunque exista un modelo de procesos definido en una organización, éste sólo representa la situación ideal en la que el equipo de desarrollo se enfrenta con problemas no previstos (excepciones). Es difícil predecir todos los imprevistos e incorporarlos en un modelo de proceso formal, por lo tanto los modelos de proceso son inevitablemente incompletos y el gestor del proyecto es responsable de modificar dinámicamente el modelo de procesos ideal para encontrar soluciones a esos problemas. Ejemplos de excepciones: Varias personas clave enferman al mismo tiempo; una avería en la computadora de seguridad deja todas las

comunicaciones fuera de servicio durante varias horas; se hace una petición no prevista de propuestas para nuevos proyectos y el esfuerzo se transfiere de un proyecto a trabajar en una propuesta.

CAMBIO EN LOS PROCESOS

El cambio en el proceso significa hacer modificaciones en el proceso existente. Se puede hacer introduciendo nuevas prácticas, métodos o herramientas, cambiando el orden de las actividades, introduciendo o eliminando entregas del proceso, o introduciendo nuevos roles y responsabilidades.
Deben fijarse metas para la mejora del proceso. Estas metas deben conducir el cambio en el proceso y, una vez realizados cambios, deben utilizarse para evaluar el progreso.

Existen cinco etapas clave en el proceso de «cambio de procesos»:

1. Identificación de la mejora. Esta etapa comprende utilizar los resultados del análisis del proceso para identificar la calidad, la duración o los cuellos de botella de los costes donde los factores del proceso influyen de forma adversa en la calidad de producto, proponiendo nuevos procedimientos, métodos y herramientas para abordar los problemas.
2. Priorización de la mejora. Esta etapa se centra en la evaluación de los cambios y en el establecimiento de las prioridades para su implementación, en base a las necesidades de mejora en áreas específicas del proceso, los costes de introducir los cambios, el impacto del cambio en la organización entre otros factores.
3. Introducción del cambio del proceso. Significa agregar nuevos procedimientos, métodos y herramientas e integrarlas con otras actividades del proceso y con los procedimientos y estándares organizacionales.
4. Capacitación en el cambio del proceso. Imponer los cambios del proceso sin la capacitación adecuada conduce a que estos cambios sean para degradar en lugar de mejorar la calidad del producto.
5. Refinamiento del cambio. Los cambios del proceso propuestos no son efectivos completamente al momento de introducirlos. Es necesario que exista una fase de ajuste donde se descubren problemas menores y se proponen e introducen las modificaciones al proceso, que pude durar varios meses.

Una vez que se introduce un cambio, el proceso de mejora se itera con análisis adicionales para identificar problemas en el proceso, proponer mejoras, etcétera.
No es práctico introducir muchos cambios al mismo tiempo pues se hace imposible evaluar el efecto de cada cambio en el proceso.
El gestor debe ser sensible a los sentimientos de la gente de su equipo cuando introduce cambios en el proceso. No hay duda de que algunas personas se sienten amenazadas por el cambio o preocupadas por perder su trabajo o ser incapaces de adaptarse a las nuevas formas de trabajar. El gestor tiene que involucrar a todo el equipo en el proceso de cambio, entendiendo sus dudas e implicándolos en la planificación del proceso nuevo.

EL MARCO DE TRABAJO PARA LA MEJORA DE PROCESOS CMMI

El software Engineering Inslitute (SEI) desarrollo el Modelo de Madurez de la Capacidad de Software del SEI (CMM) a partir de un estudio de las formas de evaluar las capacidades de los proveedores de software.
Al modelo CMM de software lo siguieron otros modelos de capacidad de madurez, entre ellos:

- El Modelo de Madurez de la Capacidad de Personal (P-CMM)
- El modelo SPICE que incluye niveles de madurez similares a los niveles del SEI, pero además identifica procesos que recorren estos niveles. A medida que subimos en nivel de madurez, el rendimiento de estos procesos clave debe mejorarse.
- El proyecto Bootstrap que utiliza los niveles de madurez del SEI, pero además incorpora:
    - Guías de calidad para ayudar en la mejora de procesos de las compañías.
    - Una distinción importante entre organización. metodología y tecnología.
    - Un modelo de proceso base (basado en el modelo utilizado por la Agencia Espacial Europea) que podría adoptarse.

En un intento de integrar la amalgama de modelos que se habían desarrollado (incluyendo sus propios modelos), el SEI se embarcó en un nuevo programa para desarrollar un modelo de capacidad integrado (CMMI).

CMMI

- El modelo CMMI intenta ser un marco de trabajo para la mejora del proceso que sea aplicable en un amplio abanico de compañías.
- 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

Los componentes del modelo son:

1. Áreas de proceso. El CMMI identifica 24 áreas de procesos que son relevantes para la capacidad y la mejora del proceso software.

2. Metas. Las metas son descripciones abstractas de un estado deseable que debería ser alcanzado por una organización. El CMMI tiene metas específicas asociadas a cada área de procesos y que definen el estado deseable para esta área. También tiene metas genéricas que son asociadas con la institucionalización de buenas prácticas.

Objetivos Genéricos: Son genéricos porque aparecen en múltiples áreas de proceso (AP).

- Cada AP tiene un solo objetivo genérico.
- Cumplir con este objetivo significa mejorar el control en la planificación e implementación de los procesos asociados con ese AP.

→El **objetivo genérico para todas las PA de nivel 2** es:

EL PROCESO ESTÁ INSTITUCIONALIZADO COMO UN PROCESO ADMINISTRADO.

Un *proceso administrado* es un proceso ejecutado que es planeado y realizado en concordancia con una política, utiliza personas capacitadas que tienen los recursos para producir salidas controladas, involucra a las personas relevantes, es monitoreado, controlado y revisado; y es evaluado por su adherencia a la descripción que se hizo de él.

→El **objetivo genérico para todas las PA de nivel 3-5**(contiene al OG de nivel 2) es:

EL PROCESO ESTÁ INSTITUCIONALIZADO COMO UN PROCESO DEFINIDO.

Un *proceso definido* es un proceso administrado que es adaptado desde un conjunto de procesos estándares de la organización, de acuerdo a una guía de adaptación de la organización; tiene una descripción de proceso que es mantenida y genera productos de trabajo, mediciones y otra información de mejora de procesos para los activos de proceso organizacionales

3. Prácticas*.* Las prácticas en el CMMI son descripciones de vías para conseguir una meta. Son actividades que aseguran que los procesos asociados con el AP serán efectivos, repetibles y duraderos. Contribuyen al cumplimiento del objetivo genérico cuando es aplicado a un AP en particular. Se pueden asociar hasta siete prácticas específicas o genéricas con cada meta dentro de cada área de procesos. Sin embargo, las organizaciones utilizan cualquier práctica apropiada para alcanzar cualquier meta del CMMI; no tienen por qué seguir las recomendaciones del CMMI.

Las metas y prácticas genéricas no son técnicas, pero están asociadas con la institucionalización de las buenas prácticas, lo que significa que dependen de la madurez de la organización. Por lo tanto, en una organización nueva que se halla en una etapa temprana del desarrollo de la madurez, la institucionalización puede significar el seguimiento de los planes y los procesos establecidos. Sin embargo, en una organización con más madurez, la institucionalización puede significar controlar los procesos utilizando técnicas estadísticas u otras técnicas cuantitativas.

→Prácticas Genéricas para Nivel 2 (bajo el Objetivo Genérico 2)

- *GP 2.1: Establecer una política organizacional*
  - Establecer y mantener una política organizacional escrita para la planificación y ejecución del proceso <x>.
- *GP 2.2: Planificar el proceso*
  - Establecer y mantener un plan para el proceso <x>.
- *GP 2.3: Proveer Recursos*
  - Proveer recursos adecuados para la ejecución del proceso <x>, desarrollando los productos de trabajo y proveyendo servicios del proceso <x>.
- *GP 2.4: Asignar Responsabilidad*
  - Asignar la responsabilidad y autoridad para ejecutar el proceso, desarrollando los productos de trabajo, y proveer los servicios del proceso <X>.
- *GP 2.5: Entrenar a las personas*
  - Entrenar a las personas para ejecutar o dar soporte al proceso <x> según sea necesario.
- *GP 2.6: Administrar configuraciones*
  - Ubicar los productos de trabajo diseñados del proceso <x>, bajo apropiados niveles de gestión de configuración.
- *GP 2.7: Identificar e involucrar a las personas relevantes.*
  - Identificar a involucrar a las personas relevantes del proceso <x> conforme se ha planeado.
- *GP 2.8: Monitorear y controlar el proceso*
  - Monitorear y controlar el proceso <x> contra el plan para realizar el proceso y tomar acciones correctivas.
- *GP 2.9: Evaluar objetivamente la adherencia.*
  - Evaluar objetivamente adherencia del proceso <x> contra su descripción, estándares y procedimientos y comunicar no conformidades.
- *GP 2.10: Revisar el estado con el nivel de administración más alto.*
  - Revisar las actividades, estados y resultados del proceso <x> con el nivel más alto de la administración y resolver los aspectos que sean necesario.

→Prácticas Genéricas para Nivel 3 (bajo el Objetivo Genérico 3)

- *GP 3.1: Establecer un proceso definido*
  - Establecer y mantener la descripción de un proceso <X> definido.
- *GP 3.2: Recolectar información de mejora*
  - Recolectar productos de trabajo, mediciones, resultados de las mediciones e información de mejora derivados de la planificación y ejecución del proceso <x> para soportar usos futuros y mejorar los procesos de la organización y los activos del proceso.

CMMI: Modelos

Constelación: es una colección de componentes CMMI que incluye un modelo, sus materiales de capacitación y sus documentos de evaluación relacionados para un área de interés.

Las constelaciones planificadas para la versión 1.2 son:

- Desarrollo
- Servicios

- Adquisición

La Constelación CMMI-DEV es la única vigente y tiene dos Modelos:

- CMMI-DEV con IPPD→ "…usado por organizaciones con implementación distribuida y que requieren integración de componentes".
- CMMI-DEV sin IPPD→ clásica

Opciones de Representación:

- Continua
- Por etapas

Representación del modelo CMMI en etapas

El modelo CMMI de niveles es comparable con el modelo CMM de software ya que provee una forma de valorar la capacidad del proceso de una organización clasificándola en uno de cinco niveles. Cada nivel de madurez tiene asociado un conjunto de áreas de proceso y metas genéricas. La mejora de procesos se lleva a cabo implementando prácticas en cada nivel, subiendo desde el nivel inferior hasta el superior del modelo.

- 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.

La ventaja del modelo CMMI en etapas, aparte de su compatibilidad con el modelo CMM, es que define un camino claro para la mejora de las organizaciones. Éstas subirán del segundo al tercer nivel, y así sucesivamente. La desventaja es que podría ser más adecuado introducir metas y prácticas correspondientes a los niveles superiores antes que las prácticas de niveles inferiores. Cuando una organización hace esto, la valoración de su madurez da una imagen engañosa.

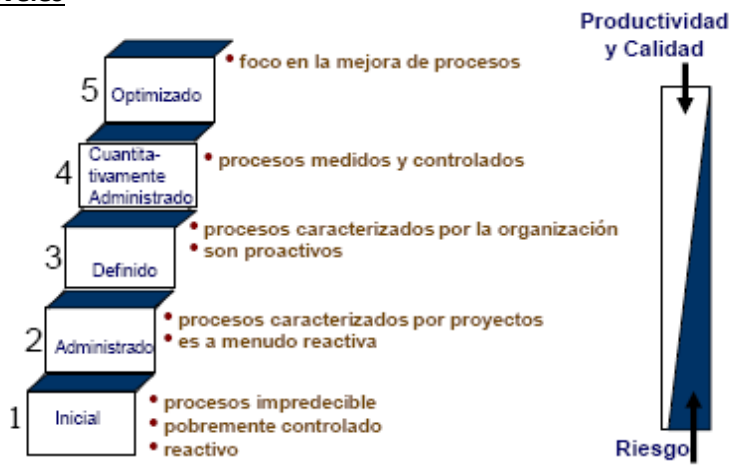
➔ "… en esta representación se ve la organización como un todo y como de hecho esta existe no puede haber un nivel cero, a diferencia de la representación continua en la cual se ve cada proceso por separado, siendo posible estar en el nivel 0 cuando un proceso no se realiza."

ORGANIZACIONES

MADURAS	• Poseen la habilidad para administrar los procesos de desarrollo y mantenimiento de software. • El proceso de desarrollo de software es comunicado a todo el personal en forma precisa y los productos de trabajo son realizados de acuerdo de procesos planeados. • Los administradores monitorean la calidad de los productos y la satisfacción del cliente. • Planificación y Presupuesto basados en performance histórica y son realistas. • Los resultados esperados pueden alcanzarse. • Los roles y responsabilidades son claramente definidos. • Los administradores monitorean la calidad de los productos de software y la satisfacción de los clientes. • Las planificaciones y presupuestos son basados en performance histórica y son realistas. • Usualmente se consiguen los resultados de costo, funcionalidad, tiempos y calidad de los productos. • Se sigue un proceso disciplinado pues todos los participantes entienden el valor de hacerlo y existe la infraestructura necesaria para darle soporte.

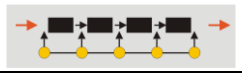
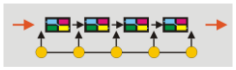
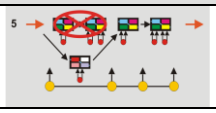
INMADURAS	• Los procesos de software generalmente son improvisados durante el curso del proyecto. • Aún si existe un proceso de desarrollo de software, no es rigurosamente aplicado. • Es reaccionaria y los administradores usualmente se concentran en resolver crisis (apagar incendios). • Planificaciones y presupuestos son excedidos debido a que no se basan en estimaciones realistas. • No hay bases para juzgar la calidad del producto.

Niveles



Visibilidad

"..Es la capacidad de saber donde estoy parado, cuanto hice y cuanto me falta, en la gestión del proceso."

Nivel 1: NO se aplican las herramientas de la ingeniería de software.	
Nivel 2: Se aplican controles básicos mediante políticas y procedimientos definidos.	
Nivel 3: Se busca que los procesos estándar estén documentados y que el proyecto se ajuste a los mismos.	
Nivel 4: En este nivel se logra que los procesos estén medidos y controlados.	
Nivel 5: Se busca la mejora continua de los procesos.	

PA por niveles

ÁREAS DEL NIVEL 2	
ÁREA	**PROPÓSITO**
Administración de Requerimientos (REQM)	Administrar los requerimientos de los productos y componentes de producto del proyecto y identificar inconsistencias entre los requerimientos y los planes y productos de trabajo del proyecto.
Planeación de Proyectos (PP)	Establecer y mantener los planes que definen las actividades del proyecto
Monitoreo y Control de Proyectos (PMC)	Provee la comprensión del progreso del proyecto para que se puedan tomar las acciones correctivas apropiadas cuando la performance del proyecto se desvía significativamente de los planes.
Administración de Acuerdo con el Proveedor (SAM)	Administrar la adquisición de productos de proveedores con los cuales existe un acuerdo formal.
Medición y Análisis (MA)	Desarrollar y sostener una capacidad de medición que sea utilizada para soportar las necesidades de información de la administración.
Aseguramiento de calidad del Producto y del Proceso (PPQA)	Proveer a la administración y al personal la adecuada comprensión respecto de los procesos y los productos de trabajo.

Administración de Configuración (CM)	Establecer y mantener la integridad de los productos de trabajo utilizando identificación de configuración, control de configuración, reporte de estado de configuración y auditorías
ÁREAS DEL NIVEL 3	
Desarrollo de Requerimientos (RD)	Producir y Analizar los requerimientos del cliente, del producto y de componentes de producto.
Solución Técnica (TS)	Diseñar, desarrollar e implementar soluciones a los requerimientos. Las soluciones, diseños e implementaciones comprenden productos, componentes de productos y ciclos de vida de procesos relacionados con los productos, tanto individuales como en combinaciones, según sea apropiado.
Integración de Producto (PI)	Ensamblar el producto desde sus componentes, asegurar que el producto integrado, funciona adecuadamente y entregar el producto.
Verificación (VER)	Asegurar que los productos de trabajo seleccionados satisfacen los requerimientos especificados.
Validación (VAL)	Demostrar que el producto o componente de producto cumple con el uso definido para él cuando es puesto en su entorno.
Foco en el Proceso Organizacional(OPF)	Implementar y planificar mejoras de proceso organizacionales basadas en la comprensión de las fortalezas y debilidades de los procesos de la organización y de los activos del proceso.
Definición del Proceso Organizacional (+ IPPD) (OPD)	Establecer y mantener un conjunto de activos del proceso y estándares de trabajo organizacional, utilizable.
Capacitación Organizacional (OT)	Desarrollar las habilidades y conocimiento de las personas de manera tal que pueda desempeñar sus roles efectiva e eficientemente.
Administración Integrada de Proyectos (IPM)	Establecer y administrar el proyecto y la participación de los involucrados relevantes de acuerdo con un proceso definido e integrado que es adaptado desde un conjunto de procesos organizacionales. También cubre el establecimiento de una visión compartida del proyecto y la estructura del equipo para equipos integrados que realizarán los objetivos del proyecto.
Administración de Riesgos (RSKM)	Identificar potenciales problemas antes de que ocurran, de manera tal que las actividades de manejo de riesgo puedan ser planeadas e invocadas como sea necesario a través del ciclo de vida del producto o proyecto, para mitigar los impactos adversos y alcanzar los objetivos.
Análisis y Resolución de Decisiones	Analizar posibles decisiones utilizando procesos de evaluación formal que evalúan alternativas identificadas contra criterios definidos.
ÁREAS DEL NIVEL 4	
Performance del Proceso Organizacional (OPP)	Establecer y mantener una comprensión cuantitativa de la performance del conjunto de procesos estándares organizacionales de apoyo a los objetivos de calidad y performance del proceso y para proveer los datos de performance del proceso, líneas base y modelos para administrar cuantitativamente los proyectos de la organización.
Gestión Cuantitativa del Proyecto (QPM)	Administrar cuantitativamente procesos definidos para el proyecto, para alcanzar la calidad establecida en el proyecto y los objetivos de performance del proceso.
ÁREAS DEL NIVEL 5	
Innovación y Desarrollo Organizacional (OID)	Seleccionar y distribuir mejorar incrementales e innovadoras que mensurablemente mejoran los procesos y tecnologías de la organización. Las mejoras soportan los objetivos de calidad de la organización y de performance del proceso como derivado de los objetivos de negocio de la organización.
Análisis Causal y Resolución (CAR)	Identificar causas de defectos y otros problemas y tomar acción para prevenir la ocurrencia de ellos en el futuro.

Objetivos y Prácticas Específicas por Áreas de Proceso

Administración de Requerimientos (REQM)

Objetivo Específico	Prácticas Específicas
Administrar Requerimientos	▪ Obtener una comprensión de los requerimientos ▪ Obtener compromisos de los requerimientos ▪ Administrar los cambios de los requerimientos ▪ Mantener rastreabilidad bidireccional de los requerimientos ▪ Identificar las inconsistencias entre el trabajo del proyecto y los requerimientos

Planeación de Proyectos (PP)

Objetivo Específico	Prácticas Específicas
Realizar Estimaciones	▪ Estimar el alcance del proyecto. ▪ Realizar estimaciones de productos de trabajo y características de las tareas. ▪ Definir el ciclo de vida del proyecto. ▪ Determinar estimaciones de esfuerzo y costo.
Desarrollar un Plan de Proyecto	▪ Establecer el presupuesto y el cronograma. ▪ Identificar los riesgos del proyecto. ▪ Planificar la Administración de los Datos. ▪ Planificar los recursos del proyecto. ▪ Planificar las necesidades de conocimiento y habilidades. ▪ Planificar la participación de los involucrados. ▪ Establecer el Plan del Proyecto.
Obtener compromiso respecto del Plan	▪ Revisar los planes que afectan al proyecto. ▪ Reconciliar el trabajo y los niveles de recursos. ▪ Obtener el compromiso del plan.

Monitoreo y Control de Proyectos (PMC)

Objetivo Específico	Prácticas Específicas
Monitorear el Proyecto contra el Plan	▪ Monitorear los parámetros de planificación del proyecto. ▪ Monitorear Compromisos. ▪ Monitorear los Riesgos del Proyecto. ▪ Monitorear la Administración de los datos. ▪ Monitorear la participación de los involucrados. ▪ Conducir Revisiones de Progreso. ▪ Conducir Revisiones de Hitos.
Administrar las acciones correctivas hasta el cierre	▪ Analizar aspectos. ▪ Tomar Acciones Correctivas ▪ Administrar Acciones Correctivas.

Administración de Acuerdo con el Proveedor (SAM)

Objetivo Específico	Prácticas Específicas
Establecer el acuerdo con los proveedores.	▪ Determinar el tipo de adquisición. ▪ Seleccionar Proveedor. ▪ Establecer acuerdos con proveedores.
Satisfacer los acuerdos con los Proveedores	▪ Ejecutar el acuerdo con proveedores. ▪ Monitorear los procesos de los proveedores seleccionados ▪ Evaluar los productos de trabajo de los proveedores seleccionados. ▪ Aceptar el Producto Adquirido ▪ Realizar la transición del producto.

Medición y Análisis (MA)

Objetivo Específico	Prácticas Específicas
Alinear las actividades de medición y análisis	▪ Establecer los objetivos de la medición. ▪ Especificar mediciones ▪ Especificar los procedimientos de almacenamiento y recolección de datos. ▪ Especificar los procedimientos de análisis.
Proveer los resultados de la medición	▪ Recolectar los datos de medición. ▪ Analizar los datos de medición. ▪ Almacenar los datos y los resultados. ▪ Comunicar los resultados.

Aseguramiento de calidad del Producto y del Proceso (PPQA)

Objetivo Específico	Prácticas Específicas
Evaluar objetivamente procesos y productos de trabajo	▪ Evaluar objetivamente los procesos. ▪ Evaluar objetivamente los servicios y productos de trabajo.
Proveer los resultados objetivamente	▪ Comunicar y Asegurar la resolución de los aspectos de no conformidad. ▪ Establecer registros.

Administración de Configuración (CM)

Objetivo Específico	Prácticas Específicas
Establecer Líneas Base	▪ Identificar ítems de configuración. ▪ Establecer un sistema de administración de configuración. ▪ Crear o Liberar Líneas Base.
Registrar y Controlar Cambios	▪ Rastrear requerimientos de cambio. ▪ Controlar ítems de configuración.
Establecer Integridad	▪ Establecer registros de administración de configuración. ▪ Realizar Auditorías de Configuración.

Desarrollo de Requerimientos (RD)

Objetivo Específico	Prácticas Específicas
Desarrollar los requerimientos del cliente	▪ Elicitar necesidades ▪ Desarrollar los requerimientos del cliente
Desarrollar los requerimientos de producto	▪ Establecer los requerimientos de productos y componentes de productos. ▪ Ubicar los requerimientos de componentes de productos. ▪ Identificar requerimientos de interfaz.
Analizar y Validar Requerimientos	▪ Establecer escenarios y conceptos operacionales. ▪ Establecer una definición de la funcionalidad requerida. ▪ Analizar los requerimientos. ▪ Analizar los requerimientos para lograr balance. ▪ Validar Requerimientos.

Solución Técnica (TS)

Objetivo Específico	Prácticas Específicas
Seleccionar Soluciones con Componentes de Productos	▪ Desarrollar Alternativas de Solución y Criterios de Selección. ▪ Seleccionar soluciones de componentes de producto.
Desarrollar el Diseño	▪ Diseñar el producto o componente de producto. ▪ Establecer el Paquete de Datos Técnico ▪ Diseñar interfaces utilizando criterio. ▪ Realizar el Análisis de Hacer, Comprar o Reusar.
Implementar el Diseño del Producto	▪ Implementar el Diseño. ▪ Desarrollar la Documentación de Soporte del Producto.

Integración de Producto (PI)

Objetivo Específico	Prácticas Específicas
Preparar para la Integración del Producto	▪ Determinar Secuencia de Integración. ▪ Establecer el Ambiente de Integración del Producto. ▪ Establecer Procedimientos y Criterio de Integración de Producto.
Asegurar Compatibilidad de Interfaces	▪ Revisar descripciones de interfaces para verificar completitud. ▪ Administrar Interfaces.
Ensamblar Componentes de Producto y Entregar el Producto	▪ Confirmar claridad para la integración de los componentes del producto. ▪ Ensamblar componentes de producto. ▪ Evaluar componentes de productos ensamblados. ▪ Empaquetar y Entregar el Producto o Componente del Producto.

Verificación (VER)

Objetivo Específico	Prácticas Específicas
Preparar para Verificación	▪ Seleccionar Productos de Trabajo para Verificación. ▪ Establecer el Ambiente para Verificación. ▪ Establecer Procedimientos y Criterios de Verificación.
Realizar Revisiones por Pares	▪ Preparar para Revisiones por Pares. ▪ Conducir las Revisiones por Pares. ▪ Analizar los datos de las Revisiones por Pares.
Verificar Productos de Trabajo Seleccionados	▪ Realizar la Verificación. ▪ Analizar los resultados de la Verificación.

Validación (VAL)

Objetivo Específico	Prácticas Específicas
Preparar para Validación	▪ Seleccionar Productos para Validación. ▪ Establecer el Ambiente para Validación. ▪ Establecer Procedimientos y Criterios de Validación.
Validar Productos o Componentes de Productos	▪ Realizar la Validación. ▪ Analizar los resultados de la Validación.

Foco en el Proceso Organizacional (OPF)

Objetivo Específico	Prácticas Específicas
Determinar Oportunidades de Mejora del Proceso	▪ Establecer Necesidades de Proceso Organizacionales. ▪ Evaluar los procesos de la organización. ▪ Identificar las mejoras de los Procesos de la Organización
Planificar e Implementar Mejoras al Proceso	▪ Establecer Planes de Acción para los Procesos. ▪ Implementar los Planes de Acción de los Procesos.
Instalar los Activos del Proceso Organizacionales Incorporar Lecciones Aprendidas	▪ Desplegar los activos de proceso organizacionales. ▪ Instalar los procesos estándares. ▪ Monitorear la instalación. ▪ Incorporar experiencias relacionadas a los procesos a los Activos de Proceso Organizacional.

Definición del Proceso Organizacional (+ IPPD) (OPD)

Objetivo Específico	Prácticas Específicas
Establecer los Activos de Proceso Organizacional	▪ Establecer Procesos Estándar. ▪ Establecer descripciones de modelos de ciclo de vida. ▪ Determinar criterios y lineamientos de adaptación. ▪ Establecer el Repositorio de Mediciones de la Organización. ▪ Establecer la Biblioteca de Activos de Procesos de la Organización.
Equilibrar las Responsabilidades de la Organización y del Equipo	▪ Establecer Reglas y Lineamientos para Equipos Integrados ▪ Establecer Mecanismos de Autoridad ▪ Permitir la Administración de IPPD

Capacitación Organizacional (OT)

Objetivo Específico	Prácticas Específicas
Establecer una Capacidad de Capacitación Organizacional	▪ Establecer necesidades de Capacitación Estratégicas. ▪ Determinar cuales Necesidades de Capacitación son la responsabilidad de la Organización. ▪ Establecer un Plan Táctico de Capacitación Organizacional. ▪ Establecer la Capacidad de Capacitación.
Proveer Capacitación Necesaria	▪ Entregar Capacitación. ▪ Establecer registros de Capacitación. ▪ Evaluar la Efectividad de la Capacitación.

Administración Integrada de Proyectos (IPM)

Objetivo Específico	Prácticas Específicas
Utilizar el Proceso Definido para el Proyecto	▪ Establecer el Proceso definido para el Proyecto. ▪ Utilizar los activos de Proceso Organizacionales para las actividades de Planificación del Proyecto. ▪ Establecer el ambiente de trabajo para el proyecto. ▪ Integrar Planes. ▪ Administrar el Proyecto utilizando Planes Integrados. ▪ Contribuir a los Activos de Proceso Organizacional.
Coordinar y Colaborar con los Involucrados Relevantes	▪ Administrar la Participación de los Involucrados. ▪ Administrar Dependencias. ▪ Resolver Aspectos de Coordinación.
Aplicar los Principios de IPPD	▪ Establecer la Visión Compartida de Proyecto. ▪ Establecer la Estructura del Equipo Integrado ▪ Asignar Requerimientos al Equipo Integrado ▪ Establecer Equipos Integrados ▪ Asegurar la Colaboración entre los Equipos Relacionados

Administración de Riesgos (RSKM)

Objetivo Específico	Prácticas Específicas
Preparar la Administración de los Riesgos	▪ Determinar Fuentes y Categorías de Riesgos ▪ Definir Parámetros de Riesgos. ▪ Establecer una estrategia de Administración de Riesgos.
Identificar y Analizar Riesgos	▪ Identificar Riesgos. ▪ Evaluar, Categorizar y Priorizar Riesgos.
Mitigar Riesgos	▪ Desarrollar Planes de Mitigación de Riesgos. ▪ Implementar Planes de Mitigación de Riesgos.

Análisis y Resolución de Decisiones (DAR)

Objetivo Específico	Prácticas Específicas
Evaluar Alternativas	▪ Establecer Lineamientos para el Análisis de Decisión. ▪ Establecer Criterios de Evaluación. ▪ Identificar Alternativas de Solución. ▪ Seleccionar Métodos de Evaluación. ▪ Evaluar Alternativas ▪ Seleccionar Soluciones

Performance del Proceso Organizacional (OPP)

Objetivo Específico	Prácticas Específicas
Establecer Líneas Base y Modelos de Desempeño (Performance)	▪ Seleccionar los procesos. ▪ Establecer medidas de performance de los procesos. ▪ Establecer Objetivos de Calidad y Performance de los Procesos. ▪ Establecer Líneas base de la Performance de los Procesos. ▪ Establecer los Modelos de Performance de los Procesos.

Gestión Cuantitativa del Proyecto (QPM)

Objetivo Específico	Prácticas Específicas
Administrar Cuantitativamente el Proyecto	▪ Establecer los Objetivos de los Proyectos. ▪ Componer el Proceso definido. ▪ Seleccionar los subprocesos que serán administrados estadísticamente. ▪ Administrar la performance del proceso.
Administrar Estadísticamente la Performance de los Subprocesos	▪ Seleccionar Técnicas Analíticas y de Medición. ▪ Aplicar Métodos Estadísticos para Comprender la variación. ▪ Monitorear la Performance de los subprocesos seleccionados. ▪ Registrar los Datos Administrados Estadísticamente.

Innovación y Desarrollo Organizacional (OID)

Objetivo Específico	Prácticas Específicas
Seleccionar Mejoras	▪ Recolectar y Analizar Propuestas de Mejora. ▪ Identificar y Analizar Innovaciones. ▪ Incorporar Mejoras en Pilotos. ▪ Seleccionar las Mejoras para Implantar.
Implantar Mejoras	▪ Planificar la Implantación. ▪ Administrar la Implantación. ▪ Medir los Efectos de la Mejora.

Análisis Causal y Resolución (CAR)

Objetivo Específico	Prácticas Específicas
Determinar las causas de los defectos	▪ Seleccionar los datos de los defectos para el Análisis. ▪ Analizar las Causas.
Direccionar las causas de los defectos	▪ Implementar Propuestas de Acción. ▪ Evaluar los efectos del cambio. ▪ Registrar Datos.

El modelo CMMI Continuo

Los modelos de madurez continuos no clasifican a las organizaciones en niveles discretos, en cambio, la valoración de la madurez es un conjunto de valores que muestran la madurez de la organización para cada proceso o grupo de procesos.
Las áreas del modelo están organizadas en cuatro grupos o categorías:

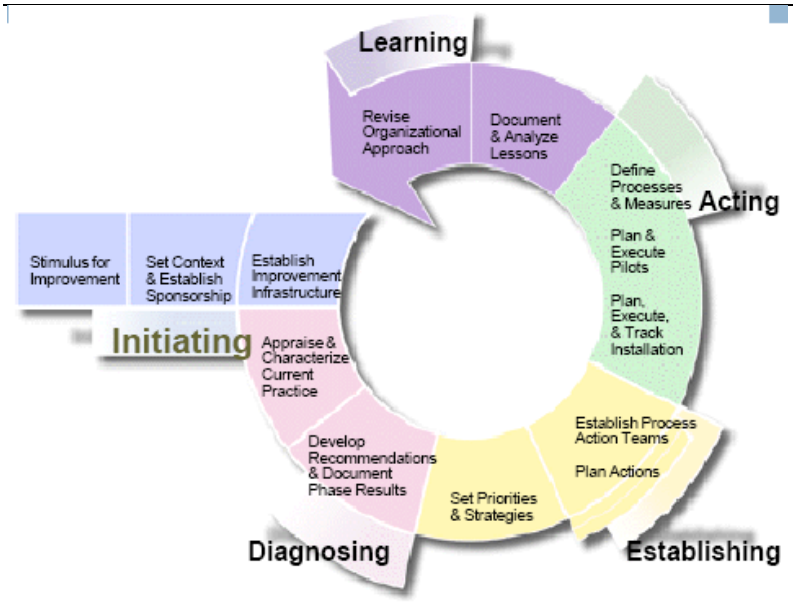
Gestión de Proceso Foco en el Proceso Organizacional Definición del Proceso Organizacional Capacitación Organizacional Performance del Proceso Organizacional Innovación y Desarrollo Organizacional	**Gestión de Proyecto** Planeamiento de Proyectos. Monitoreo y Control de Proyectos. Administración de Acuerdo con el Proveedor. Gestión Integrada de Proyectos Gestión de Riesgos Administración Cuantitativa del Proyecto
Ingeniería Administración de Requerimientos. Desarrollo de Requerimientos Solución Técnica Integración de Producto Verificación Validación	**Soporte** Medición y Análisis Aseguramiento de Calidad del Proceso y del Producto. Administración de Configuración. Análisis y Resolución de Decisiones Análisis Causal y Resolución.

El modelo CMMI continuo evalúa cada área de proceso y le asigna un nivel de valoración entre 1 y 6, según su correspondiente valoración de capacidad:

1. *No productivo.* No se satisfacen una o más de las metas específicas asociadas con el área de proceso.
2. *Productivo.* Se satisfacen las metas asociadas al área de proceso, y para todos los procesos el ámbito del trabajo a realizar es fijado y comunicado a los miembros del equipo.
3. *Gestionado.* A este nivel, las metas asociadas con el área de proceso son conocidas y tienen lugar políticas organizacionales que definen cuándo se debe utilizar cada proceso.
Debe haber planes documentados, gestión de recursos y monitorización de procedimientos a través de la institución.
4. *Definido.* Este nivel se centra en la estandarización organizacional y el desarrollo de procesos. Cada proyecto de la organización tiene un proceso de gestión creado a medida desde un conjunto de procesos organizacionales. La información y las medidas del proceso son recogidas y utilizadas para las mejoras futuras del proceso.
5. *Gestionado cuantitativamente.* En este nivel, existe una responsabilidad organizacional de usar métodos estadísticos y otros métodos cuantitativos para controlar los subprocesos. Esto significa que en el proceso de gestión debemos utilizar medidas del proceso y del producto.
6 *Optimizado.* En este nivel superior, la organización debe utilizar medidas de proceso y de producto para dirigir el proceso de mejora. Debemos analizar las tendencias y adaptar los procesos a las necesidades de los cambios del negocio.

La principal ventaja del modelo continuo es que permite más flexibilidad, es decir, las organizaciones pueden elegir procesos de mejora de acuerdo con sus propias necesidades y requerimientos. La experiencia demuestra que diferentes tipos de organizaciones tienen distintos requerimientos en su mejora de procesos. Por ejemplo, una empresa que desarrolla software para la industria aeroespacial puede centrarse en mejoras de la especificación del sistema, gestión de configuraciones y validación, mientras que una empresa de desarrollo Web estaría más interesa en los procesos de cara al cliente.

Modelo Ideal



Fase de Inicio

Propósito

- Establecer la razón del negocio.
- Alinear Sponsors y Sistemas Organizacionales.
- Establecer infraestructura y contexto para ejecutar el esfuerzo de mejora.

Beneficios de la Fase

- El esfuerzo de mejora está alineado con la estrategia y visión actual.
- El esfuerzo de mejora está basado en las necesidades reales del negocio.
- La administración está comprometida con el esfuerzo de mejora.

Resultados de la Fase

- Determinación clara del Sponsor.
- Aceptación de la mejora como una iniciativa estratégica.
- Planes detallados para desarrollar el esfuerzo de mejora.

Fase de Diagnóstico

Propósito

- Establecer línea base del proceso de madurez.
- Identificar riesgos del esfuerzo de mejora.
- Recomendar acciones específicas.

Beneficios de la Fase

- Proveer una manera ordenada de establecer prioridades.
- Fortalecer el sponsoreo.
- Proveer un marco para medición del progreso.

Resultados de la Fase

- Observaciones respecto de la Capacidad del Proceso.
- Identificación de Fortalezas y Debilidades del Proceso de Software.

- Perfil de riegos organizacional.

Fase de Establecimiento

Propósito

- Construir una infraestructura sustentable.
- Finalizar la estrategia y los planes de soporte para el mediano plazo.
- Asegurar el compromiso gerencial para proveer recursos y soporte.

Beneficios de la Fase

- Enfoque organizado y planificado de la mejora de procesos de software.
- Comprensión clara de los costo, recursos y cronogramas.
- Acciones para alcanzar la capacidad de la organización.

Resultados de la Fase

- Plan estratégico de mejora alineado con el plan estratégico de la organización.
- Distribución de los recursos para las acciones.
- Planes y actividades priorizados.

Fase de Acción

Propósito

- Efectuar los cambios en la organización.
- Administrar y rastreas la implementación de las mejoras.
- Reunir y registrar información para aprender de la mejora organizacional.

Beneficios de la Fase

- La implementación exitoso fortalece a la organización.

Resultados de la Fase

- Entregables para acciones de implementación.
- Planes para administrar la implementación

Fase de Aprendizaje

Propósito

- Completar el ciclo de mejora del proceso.
- Renovar sponsoreo para el próximo ciclo.
- Definir objetivos para el próximo ciclo.

Beneficios de la Fase

- Obtener sponsoreo continuo.
- Objetivos del próximo ciclo determinados.
- El valor del esfuerzo de mejora para la organización es cuantificado y documentado.

Resultados de la Fase

- Repositorio histórico de datos y lecciones aprendidas.
- Los actores de la mejora de procesos ganan credibilidad.
- Compromiso e Interés sostenido en el esfuerzo de mejora.

Prueba

Es el proceso destructivo de ejecutar un programa con el fin de encontrar errores

Exponer los efectos latentes ante de entregar el software. Una prueba es exitosa si provoca que el sistema se desempeñe de forma incorrecta y mostrar los errores

Principios

- Definir el resultado esperado de una prueba es importante y necesario:

Por ello un caso de prueba debe contar con: una descripción de los datos de entrada y una descripción precisa de la salida

- Un programador debe evitar probar su propio programa:

Ya que e proceso destructivo de la prueba puede ser afectado por la naturaleza psicológica de la persona

- Una empresa de programación no debería probar sus propios programas
- Inspeccionar con conciencia el resultado de la prueba

- Los casos de prueba deben ser escritos para condiciones de entradas validas y no validas
- Examinar el programa para comprobar que no hace lo que debe hacer y comprobar que hace lo que no debe hacer
- Evite los casos de prueba desechables:

La reelaboración de ellos, incurre en perdida de tiempo y recursos

- No planear un esfuerzo de prueba con la suposición de que no se encontraran errores
- La probabilidad de encontrar errores adicionales en una sección es proporcional al numero de errores ya encontrados

Si se sabe que una parte es más propensa a errores se debería probar más veces

- Las pruebas constituyen una tarea creativa

Conjunto de procedimientos y técnicas para la lectura
Inspecciones:
Son los métodos primarios, basados en el sentido común
Ambos implican que la lectura de un programa ha de ser efectuada por un equipo de personas (3 o 4), una de ellas actúa de moderador y otro es el autor del código
Donde los errores son detectados y corregidos uno a uno.
Son eficaces en errores de lógica y de codificación.

Prueba de escritorio:
Es un proceso indisciplinado donde un programador prueba su código
Auto evaluaciones comparativas:
Su objetivo no es encontrar errores, se centra más en el proceso de codificación.
Se reúnen a 6 o 20 personas, se toma código de cada uno y se hacen evaluaciones anónimas
Lista errores
-referencia datos: valores acotados, variables por regencia o por valor, valores de tipos distintos
-declaración datos: variables con nombres similares, Inicialización de objetos,
-computo: Tipos de datos distintos, cálculos mezclados, valores que exceden los límites, valores null, división por cero
-comparación: Valores boléanos, comparaciones correctas, combinaciones and y or,
-flujo control: Ramificaciones, lazos terminados
-interfaz: Nro parámetros enviados y recibidos, atributos correspondientes
-e/s: Archivos declarados, gramática, tamaños de los registros

Tipos de Pruebas

- Prueba de defectos:

Caja negra:
La persona que hace la prueba se desentiende de la estructura y comportamiento interno, su interés se dirige solamente a encontrar hechos en los cuales el programa no se comporta de acuerdo a las especificaciones.
Se ingresan combinaciones de datos generados a partir de los requerimientos y se comparan las salidas esperadas.
Las pruebas exhaustivas son imposibles de llevar a cabo ya que: no se puede probar un programa para garantizar que esta libre de errores por la cantidad de elementos a probar. Pero tomando el concepto de “economía “, el objetivo es maximizar el rendimiento
Caja blanca
Permite examinar la estructura interna del programa.
Una prueba de secuencia no garantía que el programa cumpla su especificación, podría tener secuencial faltantes o errores en los datos (tipos)
Por lo general se aplican a unidades pequeñas.
-Caso prueba:
Son especificaciones de las entradas a la prueba y de la salida esperada.

Particiones de equivalencia:

Son conjuntos de datos donde todos los objetos se procesan de igual manera.
Se identifican a partir de la especificación o documentación del usuario
Un enfoque sistemático se basa en identificarlas

- Pruebas de trayectorias:

Probar cada ejecución de la trayectoria en forma independiente respecto a todas las instrucciones condicionales, comenzando por un diagrama de flujos,

- Pruebas de integración o incrementales

Una vez probados los componentes individualmente, deben probarse el sistema parcial o completo.
Aquí es difícil encontrar el origen de los errores, y se inicia la integración de forma incremental
a. Pruebas descendentes y ascendentes.
La estrategia de pruebas descendentes y ascendentes refleja diferente enfoques de la integración del sistema. En la integración descendente, los componentes de los niveles altos de un sistema se integran y prueban antes de que se complete su diseño e implementación. En la integración ascendente, los componentes de los niveles bajos se integran y prueban antes de que se desarrollen los componentes de los niveles altos.
Las pruebas descendentes son una parte integral del proceso de desarrollo descendente en el cual este último inicia con los componentes de los niveles altos y descienden en la jerarquía de los componentes. Estos tienen la misma interfaz que los componentes pero funcionalidad muy limitada. Después de que se programa y prueba el primer componente de nivel alto, se implantan y prueban sus subcomponentes, de la misma forma. Este proceso continúa hasta que los componentes de nivel bajo se implanten. De esta forma queda completamente probado el sistema completo.
Ventaja
-si aparecen fallas grandes en lo superior
-estructura del programa permite hacer demostraciones
Problema:

- difícil armar casos de prueba para un modulo A que compruebe toda situación de B
- no se prueba completamente un modulo antes de integrar
- difícil relacionar una respuesta con el modulo probado
- difícil observar la salida

En contraste, las pruebas ascendentes comprenden integrar y probar los módulos en los niveles bajos de la jerarquía, y después asciende por la jerarquía de los módulos hasta que el módulo final se prueba. Este enfoque no requiere que el diseño arquitectónico del sistema este completo por lo que se puede comenzar en una etapa inicial del proceso de desarrollo. Se emplea donde el sistema reutiliza y modifica componentes de otros sistemas.
Ventajas:
-si aparecen fallas grandes en lo inferior
-condiciones de prueba mas fáciles de crear

- fácil observar la salida

Problemas
-requieren módulos impulsores
- no es un programa hasta que se agrega el ultimo modulo

Las pruebas de integración descendentes y ascendentes se comparan bajo cuatros encabezados:

- Validación arquitectónica: las pruebas descendentes son susceptibles a descubrir errores en la arquitectura del sistema y en el diseño de alto nivel en las etapas iniciales del proceso de desarrollo.
- Demostración del sistema: En el desarrollo descendente se dispone de un sistema funcional limitado en las primeras etapas de desarrollo.
- Implementación de las pruebas.
- Observación de las prueba: Tanto las pruebas ascendentes como descendentes pueden tener problema con la observación de la prueba. En muchos sistemas, los niveles más altos del sistema que se implementan primero en un proceso descendente no generan salidas; sin embargo, para probar estos niveles, se les debe forzar a hacerlo. El probador debe crear un entorno artificial para generar los resultados de la prueba. Para las pruebas ascendentes, también es necesario crear un

entorno artificial con el fin de que se pueda observar la ejecución de los componentes de los niveles inferiores.

- Pruebas de interfaces:

Cada módulo o subsistema tiene una interfaz que llama a otros componentes:
Tipos de errores:
-parámetros: referencia de datos
-Memoria compartida;
-paso de mensajes: cuando se solicita un servicio a un componente

- Prueba función

Es el proceso de tratar de encontrar diferencias entre el programa y su especificación externa. Que es una descripción del comportamiento del programa desde el punto de vista exterior.

- Prueba sistema

Tiene por meta comparar el programa con sus objetivos originales, lo que muestra 2 consecuencias:
1. no se limita a los sistemas
2. es imposible si no hay objetivos medibles escritos
- sentencia: si cada sentencia mencionada en los objetivos ha sido implementada
- volumen: Se lo somete a grandes volúmenes de datos, llenada hasta el límite de su capacidad. Es costosa en tiempo de maquina y personal
-esfuerzo: Se lo somete a grandes volúmenes de datos en un corto tiempo
- seguridad: protección de base de datos, archivos, memoria, usuarios, datos en los mensajes
- comportamiento: se establecen propiedades como tiempo de respuesta y velocidad bajo ciertas configuraciones
- almacenamiento: cantidades de almacenamiento principal y secundario y archivos temporales.
- configuración: pantallas, fechas, versión SO
- confiabilidad:
- recuperación: Como se recuperara el sistema de los posibles errores
- documentación: controlar exactitud y claridad, debiendo existir toda la funcionalidad provista por el sistema

- Prueba aceptación

Se compara al programa con sus requerimientos iniciales y con las necesidades actuales de los usuarios finales

- Planeamiento y control

Componentes de un plan
-objetivos
-cronograma
Se deben fijar los tiempos para cada fase además de cuando se diseñaran, escribirán y ejecutaran los casos de prueba
-responsabilidad
Se debe establecer quienes realizaran las diferentes tareas
-biblioteca de casos de prueba
-herramientas
-integración
-métodos de seguimiento
Se debe identificar los medios para seguir el progreso de las pruebas
-procedimiento de eliminación de fallos

Criterios para la terminación de pruebas

1. cuando el tiempo establecido finalizo
2. cuando todos los casos de prueba se ejecutan sin encontrar errores

Debugging
Actividad que se realiza después de ejecutar un caso de prueba exitoso. El proceso es realizado en 2 partes:
1. Determinar a naturaleza y ubicación del error

2. Corregir el error

Auditoria

La *auditoria informática* es el proceso de recoger, agrupar y evaluar evidencias para determinar si un sistema de información, mantiene la integridad de los datos, lleva a cabo eficazmente los fines de la organización, utiliza eficientemente los recursos, y cumple con las leyes y regulaciones establecidas. También permiten detectar de forma sistemática el uso de los recursos y los flujos de información dentro de una organización y determinar qué información es critica para el cumplimiento de su misión y objetivos, identificando necesidades, duplicidades, costes, valor y barreras, que obstaculizan flujos de información eficientes.

Auditoria Interna y Auditoria Externa

La auditoria 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 auditoria interna existe por expresa decisión de la Empresa, o sea, que puede optar por su disolución en cualquier momento.

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

La auditoria informática interna cuenta con algunas ventajas adicionales muy importantes respecto de la auditoria externa, las cuales no son tan perceptibles como en las auditorias convencionales. La auditoria 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 Auditorias, especialmente cuando las consecuencias de las Recomendaciones habidas benefician su trabajo.

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

¿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 auditorias 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

<u>Tipos de auditorias de calidad de software</u>

Auditorias de Proyecto

- El objetivo de esta auditoria 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 Software.
    - El diseño del producto, a medida que DDS evoluciona, satisface los requerimientos funcionales de la ERS.
    - El código es consistente con el DDS.

Auditoria Funcional

- La auditoria 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 auditoria 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.

Auditoria Física

- La auditoria 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 auditoria 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.

Metodología de Trabajo de Auditoria Informática / Proceso

El método de trabajo del auditor pasa por las siguientes etapas:

- Alcance y Objetivos.
- Estudio inicial del entorno auditable.
- Determinación de los recursos necesarios para realizar la auditoria.
- Elaboración del plan y de los Programas de Trabajo.
- Actividades propiamente dichas de la auditoria.
- Confección y redacción del Informe Final.

* Alcance y Objetivos de la Auditoria

El alcance de la auditoria expresa los límites de la misma. Debe existir un acuerdo muy preciso entre auditores y clientes sobre que se va a auditar.

Tanto los alcances como las excepciones deben figurar al comienzo del Informe Final.

Las personas que realizan la auditoria han de conocer con la mayor exactitud posible los objetivos a los que su tarea debe llegar. Deben comprender los deseos y pretensiones del cliente, de forma que las metas fijadas puedan ser cumplidas.

Una vez definidos los objetivos (objetivos específicos), éstos se añadirán a los objetivos generales y comunes de a toda auditoria Informática: La operatividad de los Sistemas y los Controles Generales de Gestión Informática.

* Estudio Inicial del entorno auditable

Para realizar dicho estudio ha de examinarse las funciones y actividades generales de la informática.

Para su realización el auditor debe conocer lo siguiente:

Recursos materiales

Es muy importante su determinación, por cuanto la mayoría de ellos son proporcionados por el cliente. Las herramientas software propias del equipo van a utilizarse igualmente en el sistema auditado, por lo que han de convenirse en lo posible las fechas y horas de uso entre el auditor y cliente.

* Los recursos materiales del auditor son de dos tipos:

a. Programas propios de la auditoria: Son muy potentes y Flexibles. Habitualmente se añaden a las ejecuciones de los procesos del cliente para verificarlos.
   Monitores: Se utilizan en función del grado de desarrollo observado en la actividad de Técnica de Sistemas del auditado y de la cantidad y calidad de los datos ya existentes.
b. Recursos materiales Software
c. Recursos materiales Hardware

Los recursos hardware que el auditor necesita son proporcionados por el cliente. Los procesos de control deben efectuarse necesariamente en las Computadoras del auditado.

Técnicas de Trabajo:	Herramientas:
- Análisis de la información recabada del auditado. - Análisis de la información propia. - Cruzamiento de las informaciones anteriores. - Entrevistas. - Simulación. - Muestreos.	- Cuestionario general inicial. - Cuestionario Checklist. - Estándares. - Monitores. - Simuladores (Generadores de datos). - Paquetes de auditoria (Generadores de Programas). - Matrices de riesgo.

* Confección y redacción del Informe Final

La función de la auditoria se materializa exclusivamente por escrito. Por lo tanto la elaboración final es el exponente de su calidad.

Resulta evidente la necesidad de redactar borradores e informes parciales previos al informe final, los que son elementos de contraste entre opinión entre auditor y auditado y que pueden descubrir fallos de apreciación en el auditor.

Estructura del informe final:

El informe comienza con la fecha de comienzo de la auditoria y la fecha de redacción del mismo. Se incluyen los nombres del equipo auditor y los nombres de todas las personas entrevistadas, con indicación de la jefatura, responsabilidad y puesto de trabajo que ostente.

Definición de objetivos y alcance de la auditoria.

Enumeración de temas considerados:

Antes de tratarlos con profundidad, se enumerarán lo más exhaustivamente posible todos los temas objeto de la auditoria.

Cuerpo expositivo:

Para cada tema, se seguirá el siguiente orden a saber:

a. Situación actual. Cuando se trate de una revisión periódica, en la que se analiza no solamente una situación sino además su evolución en el tiempo, se expondrá la situación prevista y la situación real.
b. Tendencias. Se tratarán de hallar parámetros que permitan establecer tendencias futuras.
c. Puntos débiles y amenazas.
d. Recomendaciones y planes de acción. Constituyen junto con la exposición de puntos débiles, el verdadero objetivo de la auditoria informática.
e. Redacción posterior de la Carta de Introducción o Presentación.

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